AD-A116 757   NAVAL POSTGRADUATE SCHOOL MONTEREY CA   F/G 5/8
BASELINE IMPLEMENTATIONS OF THE STANDARD LINE EDITOR (SLED).(U)
OCT 80 L A COX, R J COULTER, C F TAYLOR
UNCLASSIFIED   NPS52-80-012   NL

1 of 2

2

NPS52-80-012

NAVAL POSTGRADUATE SCHOOL

Monterey, California

Baseline Implementations of the Standard Line Editor (SLED)

L. Cox, R. Coulter,

C. Taylor, R. Burnham, and S. Smart

October 1980

Approved for public release; distribution unlimited.

Prepared for Naval Postgraduate School, Monterey
California 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral T. F. Ekelund
Superintendent

D. A. Schrady
Acting Provost

Reproduction of all or part of this report is authorized.

This report was prepared by:

Lyle A. Cox Jr.

LYLE A. COX, JR.
Assistant Professor of
Computer Science

Reviewed by:

GORDON H. BRADLEY, Chairman
Department of Computer Science

Released by:

William M. Tolles

WILLIAM M. TOLLES
Dean of Research

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-80-012 2. GOVT ACCESSION NO. AD A116757	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Baseline Implementations of the Standard Line EDitor (SLED)	5. TYPE OF REPORT & PERIOD COVERED Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Lyle Ashton Cox, Jr., et. al.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940	12. REPORT DATE October 1980
	13. NUMBER OF PAGES 117
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Text Editor
Software Engineering
Computer Networks

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In response to a recognized requirement for a more uniform man machine interface, especially in multiple machine networks, a standardized text editor was proposed (1). This editor, "SLED" was designed to be easily implementable in several commonly available higher level languages. This document reviews two baseline implementations taken directly from the SLED standards which users may want to consider when implementing SLED upon local systems. These baseline programs were written and documented with portability and understandability as goals.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Baseline Implementations of the Standard Line EDitor (SLED)

Lyle A. Cox Jr.
Editor

Department of Computer Science
Naval Postgraduate School
Monterey, California

ABSTRACT

In response to a recognized requirement for a more uniform man machine interface, especially in multiple machine networks, a standardized text editor was proposed (1). This editor, "SLED" was designed to be easily implementable in several commonly available higher level languages. This document reviews two baseline implementations taken directly from the SLED standards which users may want to consider when implementing SLED upon local systems. These baseline programs were written and documented with portability and understandability as goals.

BACKGROUND

During the Winter of 1979 and Spring of 1980, shortly after the Standard Line EDitor definition was first developed, several persons at the Naval Postgraduate School undertook implementations of SLED. Two of these implementations -- the documentation and the code -- are reproduced here.

Appendix A contains the SLED implementation developed by C. F. Taylor, Jr. on an IBM 360 system in FORTRAN. Appendix B contains an alternate implementation developed by R. M. Burnham, R. J. Coulter, and S. W. Smart in the PASCAL language.

Neither of these implementations should be considered "off the shelf" commercial quality software ready for installation. These systems do however provide two critical portions of the implementation:

1. Each contains the basic code in a "portable" higher level language, and

2. The code of each implementation is written for readability,

and the documentation has been written to allow the program to be adapted to any system with a minimum of difficulty.

It is hoped that these baseline programs will serve to facilitate the implementation of the SLED man machine interface on a variety of machines.

Acknowledgments:

I would like to thank the authors of the included codes, C. Taylor, R. Burnham, R. Coulter, and S. Smart for their particularly outstanding work on this project. I would also like to acknowledge the assistance of many of their contemporaries who attempted alternative implementations, and who helped in the evolution of the SLED standards. I would also like to thank Dr. R. W. Hamming and LTCOL. R. R. Schell for their comments and interest.

Lyle A. Cox Jr.
15 August, 1980

(1) "The Text Editor As A Uniform Man/Machine Interface. A Proposal for a Standard Editor." L. A. Cox Jr., Naval Postgraduate School Report NPS52-80-001 (Feb. 1980).

Appendix A
SLED FORTRAN Implementation
(by C. F. Taylor Jr.)

The purpose of these notes is to briefly describe the accompanying FORTRAN implementation of SLED.

SLED FORTRAN Version FORT1.1 was implemented on an IBM 360/67 computer under the CP/CMS time-sharing operating system at the W. R. Church Computer Center at the Naval Postgraduate School, in a superset of IBM FORTRAN IV, Level G, which included the "IF-THEN-ELSE" and "WHILE-DO" constructs of WATFIV-S. The addition of these two constructs greatly simplified the writing of the program and were implemented with the aid of a preprocessor written by this author. Standard FORTRAN version (the output of the preprocessor) is shown on the following pages. This version was compiled using an IBM FORTRAN IV/G compiler.

The package was implemented using the utility routines shown on pages A37-A40 of the listings, The 'TDISK EXEC" module was used to obtain the required disk space for the 4000 line (320K byte) temporary work file. It is assumed that this work space would be provided at the system level in any actual implementation of SLED. Because it is a direct-access disk file, sufficient space must be availabe in advance for the maximum capacity of the editor, which in this case is 4000 lines. This figure was selected somewhat arbitrarily; CP/CMS gives the user 800 bytes per track (IBM 2314), using the remaining space for overhead, so at 80 bytes per line, 4000 lines represents 400 tracks. Installations which do not need to edit such large files could reduce disk requirements by further limiting the capacity of the editor as follows:

1. Do a global substitution to replace the string '4000' in the program by the new capacity.

2. Modify the 'DEFINE FILE' statement in subroutine MEMORY as necessary.

3. Alter the FILEDEF statement in 'SLED1 EXEC' (or the equivalent action on another system) to request less disk space.

The basic data structures used are as follows: The file to be edited (unless a new file is being established) is read in sequentially by subroutine OPEN and stored in the work file (described above) which is conceptually a 4000 line by 80 column array. All references to the work file are made through calls to subroutine MEMORY. The lines of the work file are not necessarily kept in order. A 4000 element from the file is determined by popping the top element from the stack. This value is then recorded in the 4000 element array LPTR. LPTR(I) then always gives the address in the work file of the Ith line of the text file. As lines are deleted, their addresses are pushed back onto additions and deletions of lines by manipulating pointers rather than the text itself. Still, the work file may be accessed sequentially by using LPTR(I) as the index.

Input from the terminal is buffered in a circular queue in subroutine GETLIN. This permits the "stacking" of more than one command per line at the terminal. Calls to GETLIN return a line from the head of the queue or, if the queue is empty it reads in a new line from the terminal.

The QUIT subroutine writes the work file sequentially (using LPTR as an index) to the output file.

Additional notes which may be of interest to the local implementor follow: (including deviations from the SLED standard).

1. Integer*2 variables were used for all character storage (and for the two large arrays STACK and LPTR). Only one character was stored per word.

2. Because FORTRAN reads only fixed-length formatted records from the terminal, the carriage return cannot be used to terminate a string. This means that 'DS' and 'RS' commands must use the logical terminator character (default '$') to terminate strings and that the 'RS' command should be used as follows: RS$str1$str2$.

3. The 'RS' command replaces only the first occurrence of a string in a line because of this author's firm conviction that to allow only multiple substitution would be dangerous. In an editor such as this without a TAB function, a common string substitution would be to replace one blank with two blanks on a line. What would happen if this were done for every occurrrence of a blank on the line is too horrible to contemplate.

4. The CP/CMS operating system and IBM FORTRAN required that filenames be handled external to the program itself. The routine 'SLED EXEC' executes the simple program 'SLEDVERS' when 'SLED' is typed to alert the user to the required entry procedure, 'SLED1 <filename> <filetype>'. SLED1 EXEC then invokes the actual edit program, SLED2. The filename given the program internally is meaningless and provided only for cosmetic reasons. This system requires that only one file be opened per session.

5. Another limitation of FORTRAN required that the program read input from the line following the program prompt, not a serious problem.

SLED - STRUCTURED FORTRAN

```
C       STANDARD LINE EDITOR--FORTRAN IMPLEMENTATION
C
C       SLED VERSION FORT1.1 NPS MONTEREY 900401
C       PROGRAMMED BY:  C. F. TAYLOR, JR., CODE 55TA
C
C       FOR FURTHER INFORMATION SEE NPS TECHNICAL REPORT
C       NPS 52-80-001 BY L. A. COX, JR.
C
C
C       MAIN PROGRAM:
C
C            READS IN COMMANDS AND CALLS THE APPROPRIATE SUBROUTINE
C            IMPLEMENTS THE "EDIT" MODE
C
        COMMON /BLK1/IN,OUT
     1         /BLK5/MFLAG,ERRCT,CURLIN
        INTEGER*2 C1,C2,A,C,D,L,M,O,Q,R,S,T,V,INLINE,BLNK
        INTEGER IN,OUT,ERRCT,CURLIN,NC
        DIMENSION INLINE(80)
        LOGICAL FLAG,OPENFL,MFLAG
        DATA A/'A'/,C/'C'/,D/'D'/,L/'L'/,M/'M'/,O/'O'/,Q/'Q'/,R/'R'/,
     1     S/'S'/,T/'T'/,V/'V'/,BLNK/' '/
 2000 FORMAT (' E>')
 2010 FORMAT (' -INVALID COMMAND-  ',2A1)
 2020 FORMAT (' -NO TEXT FILE OPEN-')
 2030 FORMAT (' -ONLY ONE FILE CAN BE OPENED PER SESSION IN THIS ',
     1     'VERSION OF SLED-')
C       EDIT MODE--WRITE PROMPT
        WRITE (OUT,2000)
        CALL GETLIN(INLINE,NC)
        C1 = INLINE(1)
        C2 = INLINE(2)
C       FLAG GOES FALSE AFTER 'QUIT'
        FLAG = .TRUE.
C       OPENFL GOES TRUE AFTER FILE IS OPENED
        OPENFL = .FALSE.
C       MAIN EDIT LOOP
        WHILE (FLAG) DO
           IF (.NOT.(OPENFL .OR. (C1.EQ.O) .OR. (C1.EQ.M)
     1          .OR. (C1.EQ.V))) THEN DO
C               FILE NOT OPEN AND THIS IS NOT AN 'OPEN' COMMAND
C               OR A 'MENU' OR 'VERSION' REQUEST
                WRITE (OUT,2020)
           ELSE DO
C               EMULATE CASE (SWITCH) STATEMENT TO PROCESS COMMANDS
                IF (C1 .EQ. BLNK) GO TO 200
                IF (C1.NE.L) GO TO 10
                   CALL LIST(INLINE)
                   GO TO 200
   10           CONTINUE
                IF (C1.NE.S) GO TO 20
                   CALL SCREEN(INLINE)
                   GO TO 200
   20           CONTINUE
                IF ((C1.NE.R).OR.(C2.NE.S)) GO TO 30
```

```
                       CALL RS(INLINE)
                       GO TO 200
    30           CONTINUE
                 IF ((C1.NE.R).OR.(C2.NE.L)) GO TO 40
                       CALL RL(INLINE)
                       GO TO 200
    40           CONTINUE
                 IF ((C1.NE.A).OR.(C2.NE.L)) GO TO 50
                       CALL AL(INLINE)
                       GO TO 200
    50           CONTINUE
                 IF ((C1.NE.D).OR.(C2.NE.S)) GO TO 60
                       CALL DS(INLINE)
                       GO TO 200
    60           CONTINUE
                 IF (C1.NE.Q) GO TO 70
                       CALL QUIT
                       FLAG = .FALSE.
                       GO TO 200
    70           CONTINUE
                 IF (C1.NE.V) GO TO 80
                       CALL VERS
                       GO TO 200
    80           CONTINUE
                 IF (C1.NE.M) GO TO 90
                       CALL MENU
                       GO TO 200
    90           CONTINUE
                 IF (C1.NE.O) GO TO 100
                       IF (OPENFL) THEN DO
                             WRITE (OUT,2030)
                       ELSE DO
                             CALL OPEN
                             OPENFL = .TRUE.
                       END IF
                       GO TO 200
   100           CONTINUE
                 IF ((C1.NE.C).OR.(C2.NE.T)) GO TO 110
                       CALL CT(INLINE)
                       GO TO 200
   110           CONTINUE
C
C                IF PROGRAM GETS HERE, COMMAND IS INVALID
                       WRITE (OUT,2010) C1,C2
                       ERRCT = ERRCT + 1
C                END CASE
   200           CONTINUE
                 IF (ERRCT.GE.2) THEN DO
                       CALL MENU
                       ERRCT = 0
                 END IF
           END IF
                 IF (FLAG) THEN DO
C                      GET NEXT LINE
                       IF (.NOT.MFLAG) WRITE (OUT,2000)
```

```
            CALL GETLIN(INLINE,NC)
            C1 = INLINE(1)
            C2 = INLINE(2)
         END IF
    END WHILE
    STOP
    END

    BLOCK DATA
C
      COMMON   /BLK1/ IN,OUT
   2           /BLK3/ TFILE
   2           /BLK4/ TCHAR
   3           /BLK5/ MFLAG,ERRCT,CURLIN
      INTEGER*2 TCHAR
      INTEGER IN,OUT,TFILE,ERRCT,CURLIN
      LOGICAL MFLAG
      DATA IN/5/,OUT/6/,TCHAR/'$'/,ERRCT/0/,MFLAG/.FALSE./,CURLIN/1/,
   1     TFILE/2/
    END

    SUBROUTINE LIST(CLINE)
C
C     DISPLAYS TEXT TO THE TERMINAL
C
      CCMMON     /BLK1/ IN,OUT
   1             /BLK2/ LPTR,MAXLIN,EOF
   2             /BLK5/ MFLAG,ERRCT,CURLIN
      INTEGER*2 BLNK,LPTR,COMMA,OUTLIN,CLINE
      INTEGER IN,OUT,I,J,N1,N2,MAXLIN,EOF,ERRCT,CURLIN,FETCH
      LOGICAL MFLAG,EFLAG
      DIMENSION CLINE(80),OUTLIN(80),LPTR(4000)
      DATA BLNK/' '/,COMMA/','/,FETCH/0/
 2100 FORMAT (' -INVALID COMMAND-')
 2110 FORMAT (' ',I4,1X,80A1)
 2120 FORMAT (' -EOF-')
C
C     IF COL 2 IS BLANK, PRINT CURLIN AND EXIT
      IF (CLINE(2) .EQ. BLNK) THEN DO
         CALL MEMORY (FETCH,OUTLIN,LPTR(CURLIN))
         WRITE (OUT,2110) CURLIN, OUTLIN
      ELSE DO
C        NOW CHECK FOR LINE NUMBERS IN COMMAND
         CALL COMLIN(2,CLINE,N1,N2,EFLAG)
         IF (N1 .LE. 0) N1 = 1
         IF (N1 .GT. N2) EFLAG = .TRUE.
         IF (N1 .GE. EOF) EFLAG = .TRUE.
         IF (EFLAG) THEN DO
            ERRCT = ERRCT + 1
            WRITE (OUT,2100)
         ELSE DO
            ERRCT = 0
            I = N1
```

```
                   WHILE ((I.LE.N2).AND.(I.LT.EOF)) DO
                        CALL MEMORY (FETCH,OUTLIN,LPTR(I))
                        WRITE (OUT,2110) I, OUTLIN
                        CURLIN = I
                        I = I + 1
                   END WHILE
                   IF (I.GE.EOF) WRITE (OUT,2120)
              END IF
         END IF
      RETURN
      END


      SUBROUTINE SCREEN(CLINE)
C
C        DISPLAYS 20 LINES BEGINNING WITH CURLIN OR OTHER SPECIFIED LINE
C
         COMMON     /BLK1/ IN,OUT
     1              /BLK2/ LPTR,MAXLIN,EOF
     2              /BLK5/ MFLAG,ERRCT,CURLIN
         INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,I,N1,N2,LIMIT,
     1        FETCH,N
         INTEGER*2 CLINE,LPTR,OUTLIN,BLNK
         LOGICAL MFLAG,EFLAG
         DIMENSION CLINE(80),OUTLIN(80),LPTR(4000)
         DATA BLNK/' '/,FETCH/0/
 2100    FORMAT (' -INVALID COMMAND-')
 2110    FORMAT (' ',I4,1X,80A1)
 2120    FORMAT (' -EOF-')
C
         EFLAG = .FALSE.
C        FIND OUT WHETHER USER SPECIFIED A LINE
         IF (CLINE(2).NE.BLNK) THEN DO
              CALL COMLIN(2,CLINE,N1,N2,EFLAG)
              IF (N1 .LE. 0) N1 = 1
              IF (N1 .GE. EOF) EFLAG = .TRUE.
              IF (.NOT..EFLAG) CURLIN = N1
         END IF
         IF (EFLAG) THEN DO
              ERRCT = ERRCT + 1
              WRITE (OUT,2100)
         ELSE DO
              ERRCT = 0
              LIMIT = MIN0(CURLIN+19,EOF-1)
              DO 10 I = CURLIN,LIMIT
                   CALL MEMORY(FETCH,OUTLIN,LPTR(I))
                   WRITE (OUT,2110) I,OUTLIN
   10         CONTINUE
              CURLIN = LIMIT
              IF (LIMIT .EQ. EOF-1) WRITE (OUT,2120)
         END IF
      RETURN
      END
      SUBROUTINE RL(CLINE)
C
C        REPLACES CURRENT LINE OR THE SPECIFIED LINE OR LINES WITH
C        ANY NUMBER OF LINES
C
         COMMON    /BLK1/ IN,OUT
     1             /BLK2/ LPTR,MAXLIN,EOF
     2             /BLK5/ MFLAG,ERRCT,CURLIN
         INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,N,N1,N2,I,J,LIMIT,
     1       STORE
         INTEGER*2 CLINE,LPTR,BLNK
         LOGICAL MFLAG,EFLAG
         DIMENSION CLINE(80),LPTR(4000)
         DATA BLNK/' '/,STORE/1/
 2100    FORMAT (' -INVALID COMMAND-')
C
         N1 = CURLIN
         N2 = N1
C        DETERMINE WHICH LINE(S) TO REPLACE
         IF (CLINE(3).NE.BLNK) THEN DO
             CALL COMLIN(3,CLINE,N1,N2,EFLAG)
             IF (N1 .LE. 0) EFLAG = .TRUE.
             IF (N1 .GE. EOF) EFLAG = .TRUE.
             IF (.NOT. EFLAG) CURLIN = N1
         END IF
         IF (EFLAG) THEN DO
             ERRCT = ERRCT + 1
             WRITE (OUT,2100)
         ELSE DO
             ERRCT = 0
             IF (N2 .GE. EOF) N2 = EOF - 1
C            REMOVE DESIGNATED LINES
             N = N2 - N1 + 1
             DO 20 I = 1,N
                 LIMIT = EOF - 2
                 CALL PUSH(LPTR(N1))
                 DO 10 J = N1,LIMIT
                     LPTR(J) = LPTR(J+1)
   10            CONTINUE
                 EOF = EOF - 1
   20        CONTINUE
C            NOW INPUT REPLACEMENT LINES
             CALL INPUT
         END IF
       RETURN
       END

       SUBROUTINE AL(CLINE)
C
C          INPUT TEXT AFTER LINE N
C
           COMMON    /BLK1/ IN,OUT
     1             /BLK5/ MFLAG,ERRCT,CURLIN
      INTEGER*2 BLNK,CLINE
      INTEGER IN,OUT,I,J,N,N1,N2,ERRCT,CURLIN
      LOGICAL MFLAG,EFLAG
      DIMENSION CLINE(80)
      DATA BLNK/' '/
 2100 FORMAT (' -INVALID COMMAND-')
C
C     EXTRACT LINE NUMBER FROM COMMAND LINE
      CALL COMLIN(3,CLINE,N1,N2,EFLAG)
      IF (N1 .LT. 0) EFLAG = .TRUE.
      N = N1
      IF (EFLAG) THEN DO
          ERRCT = ERRCT + 1
          WRITE (OUT,2100)
      ELSE DO
          ERRCT = 0
          CURLIN = N + 1
          CALL INPUT
      END IF
   RETURN
   END


   SUBROUTINE DS(CLINE)
C
C     PART OF SLED PACKAGE
C     DISPLAYS ALL LINES CONTAINING THE DESIGNATED STRING, POSSIBLY
C     LIMITED TO LINES N THROUGH M.
C
      COMMON /BLK1/ IN,OUT
     1       /BLK2/ LPTR,MAXLIN,EOF
     2       /BLK5/ MFLAG,ERRCT,CURLIN
      INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,N1,N2,NC,FETCH,MC1
      INTEGER*2 CLINE,LPTR,BLNK,STRING,LINE
      LOGICAL MFLAG,EFLAG,MATCH,FOUND
      DIMENSION CLINE(80),LINE(80),LPTR(4000),STRING(80)
      DATA BLNK/' '/,FETCH/0/
 2100 FORMAT (' -INVALID COMMAND-')
 2110 FORMAT (' ',I4,1X,80A1)
 2220 FORMAT (' OLD STRING?>')
 2250 FORMAT (' -NO STRING FOUND-')
C
C     DEFAULT VALUES
      FOUND = .FALSE.
      EFLAG = .FALSE.
      N1 = 1
      N2 = EOF - 1
C     DETERMINE WHETHER N1,N2 WERE SPECIFIED BY USER
      IF (CLINE(3) .NE. BLNK) THEN DO
          CALL COMLIN(3,CLINE,N1,N2,EFLAG)
          IF (N1 .GE. EOF) EFLAG = .TRUE.
          IF (N2 .GE. EOF) N2 = EOF - 1
          IF (N1 .LE. 0) N1 = 1
      END IF
```

```
        IF (EFLAG) THEN DO
            ERRCT = ERRCT + 1
            WRITE (OUT,2100)
        ELSE DO
            ERRCT = 0
C           FETCH STRING; ISSUE PROMPT IF NECESSARY
            IF (.NOT. MFLAG) WRITE (OUT,2220)
            CALL GETLIN (STRING,NC)
            IF (NC .LE. 0) THEN DO
                ERRCT = ERRCT + 1
                WRITE (OUT,2100)
            ELSE DO
                DO 20 I = N1,N2
                    CALL MEMORY(FETCH,LINE,LPTR(I))
                    CALL SEARCH(LINE,STRING,NC,MATCH,MC1)
                    IF (MATCH) THEN DO
                        FOUND = .TRUE.
                        WRITE (OUT,2110) I,LINE
                        CURLIN = I
                    END IF
  20            CONTINUE
                IF (.NOT. FOUND) WRITE (OUT,2250)
            END IF
        END IF
      RETURN
      END
      SUBROUTINE RS(CLINE)
C
C       PART OF SLED PACKAGE
C       REPLACES THE FIRST OCCURRENCE OF STRING1 WITH STRING2 ON THE
C       CURRENT LINE OR WITHIN THE SPECIFIED RANGE OF LINES
C
        COMMON /BLK1/ IN,OUT
     1        /BLK2/ LPTR,MAXLIN,EOF
     2        /BLK5/ MFLAG,ERRCT,CURLIN
        INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,N1,N2,I,J,K,L,M,
     1        N,MC1,NC1,NC2,FETCH,STORE
        INTEGER*2 CLINE,LPTR,BLNK,STR1,STR2,LINE
        DIMENSION LPTR(4000),CLINE(80),STR1(80),STR2(80),LINE(80)
        DATA BLNK/' '/,FETCH/0/,STORE/1/
        LOGICAL MFLAG,FOUND,MATCH,EFLAG
 2100   FORMAT (' -INVALID COMMAND-')
 2110   FORMAT (' ',I4,1X,80A1)
 2230   FORMAT (' OLD STRING?>')
 2240   FORMAT (' NEW STRING?>')
 2250   FORMAT (' -NO STRING FOUND-')
C
C       DEFAULT CONDITIONS
        N1 = CURLIN
        N2 = N1
        EFLAG = .FALSE.
C       INTERPRET COMMAND LINE
        IF (CLINE(3).NE.BLNK) THEN DO
            CALL COMLIN(3,CLINE,N1,N2,EFLAG)
            IF (N1 .LE. 0) EFLAG = .TRUE.
```

```
              IF (N1 .GE. EOF) EFLAG = .TRUE.
              IF (N2 .GE. EOF) N2 = EOF - 1
         END IF
         IF (EFLAG) THEN DO
              ERRCT = ERRCT + 1
              WRITE (OUT,2100)
         ELSE DO
              ERRCT = 0
C             READ IN TWO STRINGS; PROMPT IF NECESSARY
              IF (.NOT. MFLAG) WRITE (OUT,2230)
              CALL GETLIN(STR1,NC1)
              IF (NC1 .LE. 0) EFLAG = .TRUE.
              IF (.NOT. MFLAG) WRITE (OUT,2240)
              CALL GETLIN(STR2,NC2)
              IF (EFLAG) THEN DO
                   ERRCT = ERRCT + 1
                   WRITE (OUT,2100)
                   RETURN
              END IF
C             NOW FIND STRING1
              FOUND = .FALSE.
              DO 50 K = N1,N2
                   CALL MEMORY(FETCH,LINE,LPTR(K))
                   CALL SEARCH(LINE,STR1,NC1,MATCH,MC1)
                   IF (MATCH) THEN DO
C                       NOW MAKE SUBSTITUTION
                        J = MC1
                        FOUND = .TRUE.
C                       DELETE STRING1
                        DO 20 I = 1,NC1
                             DO 10 M = J,79
                                LINE(M) = LINE(M+1)
   10                        CONTINUE
   20                   CONTINUE
C                       NOW MAKE ROOM FOR STRING2
                        IF (NC2 .GT. 0) THEN DO
                             DO 40 L = 1,NC2
                                M = 81 - J
                                DO 30 I = 2,M
                                   LINE(82-I) = LINE(81-I)
   30                           CONTINUE
   40                        CONTINUE
C                            NOW INSERT NEW STRING
                             DO 45 I = 1,NC2
                                LINE(J+I-1) = STR2(I)
   45                        CONTINUE
                        END IF
C                       STORE REVISED LINE
                        CALL MEMORY(STORE,LINE,LPTR(K))
C                       DISPLAY REVISED LINE
C                       WRITE (OUT,2110) K,LINE
C                       REMOVE "C" IN CC 1 OF ABOVE LINE TO ENABLE
C                       DISPLAY OF EACH LINE IN WHICH A STRING HAS BEEN
C                       REPLACED
                        CURLIN = K
```

```
                     END IF
   50            CONTINUE
                 IF (.NOT. FOUND) WRITE (OUT,2250)
            END IF
        RETURN
        END
        SUBROUTINE CT
C
C           PART OF SLED PACKAGE
C           CHANGES THE MESSAGE TERMINATOR TO ANY VALID CHARACTER
C
            COMMON /BLK1/  INT,OUT
     1             /BLK4/  TCHAR
     2             /BLK5/  MFLAG,ERRCT,CURLIN
            INTEGER IN,OUT,ERRCT,CURLIN,NC
            INTEGER*2 TCHAR,INLIN,BLNK
            LOGICAL MFLAG
            DIMENSION INLIN(80)
            DATA BLNK/' '/
 2100       FORMAT (' -INVALID COMMAND-')
 2200       FORMAT (' TERMINATOR?>')
C
            IF (.NOT. MFLAG) THEN DO
C                ISSUE PROMPT
                 WRITE (OUT,2200)
            END IF
            CALL GETLIN(INLIN,NC)
            IF ((NC.EQ.0).OR.(INLIN(1).EQ.BLNK)) THEN DO
                 ERRCT = ERRCT + 1
                 WRITE (OUT,2100)
            ELSE DO
                 TCHAR = INLIN(1)
            END IF
        RETURN
        END
        SUBROUTINE MENU
C
C           PART OF SLED PACKAGE
C           PROVIDES USER WITH A SUMMARY OF AVAILABLE COMMANDS
C           AND THEIR FORMATS.
C
        COMMON /BLK1/ IN,OUT
        INTEGER IN,OUT
  200   FORMAT (' SLED COMMAND SUMMARY:'//' LINE/TEXT INSERT',T39,
     1     'STRING REPLACEMENT'/3X,'ALN',T10,'INSERT <A>FTER <L>INE N',
     2     T40,'RS$P$Q$',T48,'<R>EPLACE <S>TRING',/3X,'RLN',T10,
     3     '<R>EPLACE <L>INE N .OR.',T40,'RSN$P$Q$',T50,'"P" WITH "Q" IN'/
     4     3X,'RLN,M',T15,'LINES N THRU M',T40,'RSN,M$P$Q$',T52,
     5     'INDICATED LINES.'/' OUTPUT COMMANDS',T38,'STRING SEARCH'/
     6     3X,'L',T10,'DISPLAY CURRENT <L>INE',T40,'DS$P$',T48,
     7     '<D>ISPLAY LINES '/3X,'LN',T10,'OR LINE N.',T50,
     8     'WITH <S>TRING "P",'/3X,'LN,M',T10,'LINES N THRU M',T40,
     9     'DSN,M$P$',T48,'OR SHOW ANY LINES')
  201 FORMAT (3X,'S',T10,'<S>HOW A ',
     X     '"SCREEN" OF LINES',T50,'N-M CONTAINING "P"'/3X,'SN',T10,
     1    'SHOW A SCREEN FROM LINE N',T38,'CONTROL COMMANDS'/
     2    3X,'M',T11,'SHOW COMMAND <M>ENU (THIS)',T40,'Q',T48,
     3    '<O>PEN A FILE OR'/3X,'V',T10,'SHOW <V>ERSION INFORMATION',
     4    T52,'CREATE A FILE FOR EDITING'/T40,'CT  <C>HANGE THE LOGICAL'/
     5    5X,'TO <Q>UIT THE EDIT TYPE "Q<RET>"',T40,'MESSAGE <T>ERMIN',
     6    'ATOR')
      WRITE (OUT,200)
      WRITE (OUT,201)
      RETURN
      END
      SUBROUTINE VERS
C
C         PAPT OF SLED PACKAGE
C
      COMMON /BLK1/ IN,OUT
      INTEGER IN,OUT
  220 FORMAT (' SLED VERSION FORT1.1 NPS MONTEREY 800401'/
     1    ' LOCAL EXPERT IS C. TAYLOR 408-646-2691 0800-1700 PST/PDT'//
     2    ' LINE DELETE KEY IS < > (ASCII) OR <CENT SIGN>  ',
     3    '(EBCDIC)'/' CHARACTER DELETE KEY IS <@>'//
     4    ' EDITOR LOGICAL MESSAGE TERMINATORS ARE:'/
     5    '      (1) <RETURN>    AND     (2) <$>'//
     6    ' AND CAN BE CHANGED TO ANY STANDARD FORTRAN CHARACTER.'/
     7    ' ALL INPUT IS TRANSLATED TO UPPER CASE.'/
     8    ' THE FOLLOWING DEVIATIONS FROM SLED STANDARD WERE REQUIRED:')
  221 FORMAT (
     9    ' (1)   THE UNIVERSAL ENTRY COMMAND "SLED" INVOKES INSTRUCTIONS'
     *    /'       FOR A NON-STANDARD ENTRY:  "SLED1 <FILENAME> <FILETYPE>
     1"'/        ' (2)  ONLY ONE FILE PER SESSION CAN BE OPENED.'/
     2    ' (3)   MAXIMUM FILESIZE IS 4000 LINES.'/
     3    ' (4)   THE USER IS ASKED TO INDICATE WHETHER HE IS EDITING A '/
     4    '       NEW FILE IN ORDER TO PREVENT A DISK READ ERROR IN FORTRA
     5N.'/' (5)   WHEN <RETURN> IS USED AS A LOGICAL MESSAGE '/
     6    '       TERMINATOR, THE LINE OR STRING IS PADDED WITH BLANKS UN'
     7/   '       THE RIGHT.  THIS AFFECTS THE RS FUNCTION ONLY.')
  222 FORMAT (
     1    ' (6)   ONLY THE FIRST OCCURRENCE OF A STRING IN EACH LINE'/
     2    '       IS REPLACED TO PERMIT FREE SUBSTITUTIONS OF BLANKS')
      WRITE (OUT,220)
      WRITE (OUT,221)
      WRITE (OUT,222)
      RETURN
      END
      SUBROUTINE OPEN
C
C         OPENS TEXT FILE AND WORKSPACE FILE IF IT ALREADY EXISTS
C         READS TEXT FILE INTO WORKSPACE
C         INITIALIZES POINTERS ETC.
C
      COMMON /BLK1/ IN,OUT
     1       /BLK2/ LPTR,MAXLIN,EOF
     2       /BLK3/ TFILE
     3       /BLK5/ MFLAG,ERRCT,CURLIN
     4       /BLK6/ STACK,STKPTR
      INTEGER IN,OUT,TFILE,LINE,ERRCT,CURLIN,STORE,MAXLIN,EOF,
     1         STKPTR,I,NC
         INTEGER*2 LPTR,STACK,FNAME,INLIN,YES,NO,REPLY,BLINE
         LOGICAL MFLAG
         DIMENSION LPTR(4000),BLINE(80),STACK(4000),FNAME(80),INLIN(80)
         DATA YES/'Y'/,BLINE/80*' '/,NO/'N'/,STORE/1/
 1000    FORMAT (80A1)
 2040    FORMAT (' -',I4,' LINES IN FILE:  ',80A1)
 2050    FORMAT (' -CREATING FILE:  ',80A1)
 2400    FORMAT (' FILENAME?>')
 2410    FORMAT (' IS THIS A NEW FILE?>')
 2420    FORMAT (' -MAX CAPACITY 4000 LINES EXCEEDED-')
C
C        INITIALIZE
         MAXLIN = 4000
         CURLIN = 1
C        READ IN FILENAME (COSMETIC)
         IF (.NOT. MFLAG) WRITE (OUT,2400)
         CALL GETLIN(FNAME,NC)
C        ASK WHETHER IT IS A NEW FILE (TO PREVENT FORTRAN READ ERROR)
    5    WRITE (OUT,2410)
         READ (IN,1000) INLIN
         REPLY = INLIN(1)
         IF ((REPLY.NE.YES).AND.(REPLY.NE.NO)) GO TO 5
         IF (REPLY.EQ.YES) THEN DO
              WRITE (OUT,2050) FNAME
              EOF = 1
              STKPTR = 1
C             ACTIVATE FILE WITH AN ACCESS
              LPTR(1) = 1
              CALL MEMORY(STORE,BLINE,LPTR(1))
         ELSE DO
C             NOW READ IN TEXT FILE
              LINE = 0
              WHILE (.TRUE.) DO
                   READ (TFILE,1000,END=10) INLIN
                   LPTR(LINE+1) = LINE + 1
                   CALL MEMORY(STORE,INLIN,LPTR(LINE+1))
                   LINE = LINE + 1
              END WHILE
   10         CONTINUE
              IF (LINE.GE.MAXLIN) THEN DO
                   WRITE (OUT,2420)
                   STOP
              END IF
              STKPTR = LINE + 1
              EOF = STKPTR
C             TELL USER FILE OPEN
              WRITE (OUT,2040) LINE,FNAME
         END IF
         DO 20 I = 1,MAXLIN
              STACK(I) = I
   20    CONTINUE
      RETURN
      END
      SUBROUTINE QUIT
```

```
C
C        PART OF THE SLED PACKAGE
C        CLOSES OUT THE WORK FILE AND WRITES THE NEW OR UPDATED
C        TEXT FILE
C
         COMMON /BLK2/ LPTR,MAXLIN,EOF
     1          /BLK3/ TFILE
     2          /BLK1/ IN,OUT
         INTEGER MAXLIN,IN,OUT,EOF,TFILE,L,LIMIT
         INTEGER*2 LINE,LPTR
         DIMENSION LPTR(4000),LINE(80)
 2000    FORMAT (80A1)
 2450    FORMAT (' -',I4,' LINES WRITTEN-')
C
         REWIND TFILE
         LIMIT = EOF - 1
         DO 90 L = 1,LIMIT
              CALL MEMORY(FETCH,LINE,LPTR(L))
              WRITE (TFILE,2000) LINE
   90    CONTINUE
         WRITE (OUT,2450) LIMIT
      RETURN
      END
      SUBROUTINE MEMORY(ACTION,LINE,PTR2)
C
C        PART OF SLED PACKAGE
C        HANDLES ALL MEMORY REFERENCES USING DIRECT-ACCESS DISK FILE
C        CURRENT CAPACITY IS 4000 LINES
C        REQUIRES AT LEAST 3 DEDICATED CYLINDERS OF DISK SPACE FOR
C        WORK FILE UNDER CP/CMS ON AN IBM 360/67
C
         COMMON /BLK3/ TFILE
         INTEGER WFILE,TFILE,ACTION,STORE,PTR,AVAR,ERRS
         INTEGER*2 LINE,PTR2
         DIMENSION LINE(80)
         DATA STORE/1/
 1000    FORMAT (80A1)
C
C        DEFINE WORK FILE
         WFILE = 13
         DEFINE FILE 13(4000,80,E,AVAR)
C        CONVERT PTR2 FROM INTEGER*2 TO INTEGER
         PTR = PTR2
C        INITIALIZE READ ERROR COUNTER AND BEGIN
         ERRS = 0
         IF (ACTION .EQ. STORE) THEN DO
              WRITE (WFILE'PTR,1000) LINE
         ELSE DO
C             FETCH
    5         READ (WFILE'PTR,1000,ERR=99) LINE
         END IF
      RETURN
   99    ERRS = ERRS + 1
         IF (ERRS .LT. 10) GO TO 5
         STOP
      END
      SUBROUTINE INPUT
C
C        IMPLEMENTS THE INPUT MODE
C
         COMMON /BLK1/ IN,OUT
     1          /BLK2/ LPTR,MAXLIN,EOF
     2          /BLK5/ MFLAG,ERRCT,CURLIN
         INTEGER*2 PD,LPTR,BLNK,OUTLIN
         INTEGER MAXLIN,EOF,ERRCT,CURLIN,STORE,I,J,IN,OUT,NC
         LOGICAL MFLAG
         DIMENSION LPTR(4000),OUTLIN(80)
         DATA PD/'.'/,STORE/1/,BLNK/' '/
 2110    FORMAT (' I>')
C        IF NO INPUT IN QUEUE, PROMPT USER
         IF (.NOT.MFLAG) WRITE (OUT,2110)
         CALL GETLIN(OUTLIN,NC)
         WHILE (.NOT.((OUTLIN(1).EQ.PD).AND.(OUTLIN(2).EQ.BLNK))) DO
           IF (NC .GT. 0) THEN DO
C              UNLESS IT WAS A NULL LINE
C              MAKE ROOM FOR NEW INPUT
               IF (CURLIN.LT.EOF) THEN DO
                    J = EOF - CURLIN
                    DO 10 I = 1,J
                         LPTR(EOF + 1 - I) = LPTR(EOF - I)
   10               CONTINUE
               ELSE DO
                    CURLIN = EOF
C                   KEEPS INPUT TEXT CONTIGUOUS
               END IF
               EOF = EOF + 1
C              GET A NUMBER FOR NEW LINE FROM STACK
               CALL POP(LPTR(CURLIN))
C              NOW STORE THE NEW LINE
               CALL MEMORY(STORE,OUTLIN,LPTR(CURLIN))
               CURLIN = CURLIN + 1
            END IF
C           IF NOTHING IN QUEUE, PROMPT USER
            IF (.NOT.MFLAG) WRITE (OUT,2110)
            CALL GETLIN(OUTLIN,NC)
         END WHILE
      RETURN
      END

      SUBROUTINE GETLIN(OUTLIN,NC)
C
C        GETS A LINE FROM TERMINAL; QUEUES UP MULTIPLE LINES
C
         COMMON    /BLK1/ IN,OUT
     1             /BLK4/ TCHAR
     2             /BLK5/ MFLAG,ERRCT,CURLIN
         INTEGER*2 INLIN,OUTLIN,QUEUE,BLNK,TCHAR
         INTEGER IN,OUT,ERRCT,CURLIN,I,J,K,LINELN,JJ,ENDJ,NC,NCHARS
         LOGICAL MFLAG,NFLAG
         DIMENSION INLIN(80),OUTLIN(80),QUEUE(80,10),NCHARS(10)
         DATA BLNK/' '/,LINELN/80/,BQ/0/,ENDQ/0/
 1010    FORMAT (80A1)
 2060    FORMAT (' -TRUNCATED; ONLY 10 ITEMS PER LINE-')
 2070    FORMAT (' -ILLEGAL CHARACTER OR BLANK COMMAND-')
C
C        MFLAG GOES TRUE WHEN MULTIPLE INPUT LINES ARE STACKED
         IF (.NOT.MFLAG) THEN DO
              READ (IN,1010,ERR=99,END=98) INLIN
              I = 1
              WHILE ((I.LE.LINELN).AND.(INLIN(I).NE.TCHAR)) DO
                   OUTLIN(I) = INLIN(I)
                   I = I + 1
              END WHILE
              NC = I - 1
              IF (INLIN(I).EQ.TCHAR) MFLAG = .TRUE.
              IF (I.LE.LINELN) THEN DO
                   DO 20 K = I,LINELN
                        OUTLIN(K) = BLNK
   20              CONTINUE
              END IF
              WHILE (INLIN(I).EQ.TCHAR) DO
                   IF (ENDQ.GE.10) THEN DO
                        WRITE (OUT,2060)
                        INLIN(I) = BLNK
                   ELSE DO
                        ENDQ = ENDQ + 1
                        I = I + 1
                        J = 1
                        NFLAG = .TRUE.
                        WHILE ((I.LE.LINELN).AND.(INLIN(I).NE.TCHAR)) DO
                             QUEUE(J,ENDQ) = INLIN(I)
                             IF (INLIN(I).NE.BLNK) NFLAG = .FALSE.
                             I = I + 1
                             J = J + 1
                        END WHILE
                        NCHARS(ENDQ) = J - 1
                        IF((I.GT.LINELN).AND.NFLAG) NCHARS(ENDQ)=0
                        IF (J.LE.LINELN) THEN DO
                             DO 30 K = J,LINELN
                                  QUEUE(K,ENDQ) = BLNK
   30                        CONTINUE
                        END IF
                   END IF
              END WHILE
         ELSE DO
C             GET LINE FROM QUEUE INSTEAD
              BQ = BQ + 1
              NC = NCHARS(BQ)
              DO 40 I = 1,LINELN
                   OUTLIN(I) = QUEUE(I,BQ)
   40         CONTINUE
              IF (BQ.EQ.ENDQ) THEN DO
                   BQ = 0
                   ENDQ = 0
```

```
                    MFLAG = .FALSE.
                 END IF
            END IF
         RETURN
   98       CONTINUE
            REWIND IN
   99       CONTINUE
            WRITE (OUT,2070)
            OUTLIN(1) = BLNK
         RETURN
         END

         SUBROUTINE PUSH(X)
C
C           PUSHES A POINTER TO A FREE LINE ONTO THE STACK
C
            COMMON     /BLK1/ IN,OUT
     1                 /BLK5/ STACK,STKPTR
            INTEGER STKPTR,IN,OUT
            INTEGER*2 STACK,X
            DIMENSION STACK(4000)
 2080       FORMAT (' -FREE LINE LIST STACK OVERFLOW-')
C
            IF (STKPTR.GT.1) THEN DO
                 STKPTR = STKPTR - 1
                 STACK(STKPTR) = X
            ELSE DO
C                STACK OVERFLOW
                 WRITE (OUT,2080)
            END IF
         RETURN
         END

         SUBROUTINE POP(X)
C
C           POPS A POINTER TO A FREE LINE FROM THE STACK
C
            COMMON     /BLK1/ IN,OUT
     1                 /BLK2/ LPTR,MAXLIN,EOF
     2                 /BLK6/ STACK,STKPTR
            INTEGER STKPTR,MAXLIN,EOF,IN,OUT
            INTEGER*2 STACK,LPTR,X
            DIMENSION STACK(4000),LPTR(4000)
 2090       FORMAT (' -ALL SYSTEM BUFFERS FULL-')
            X = STACK(STKPTR)
            IF (STKPTR .LT. MAXLIN) THEN DO
                 STKPTR = STKPTR + 1
            ELSE DO
                 WRITE (OUT,2090)
            END IF
         RETURN
         END
```

```
      SUBROUTINE CNVRT(STRING,I,J,N)
C
C        CONVERTS CHARACTERS I THROUGH J OF STRING INTO AN INTEGER N
C
         INTEGER*2 STRING,DIGIT
         INTEGER I,J,N,K,L
         DIMENSION STRING(80),DIGIT(10)
         DATA DIGIT/'0','1','2','3','4','5','6','7','8','9'/
         N = 0
         DO 20 K = I,J
             L = 1
             WHILE (STRING(K).NE.DIGIT(L)) DO
                 L = L + 1
             END WHILE
             IF (L .LE. 10) THEN DO
                 N = N + (L-1)*(10**(J-K))
             ELSE DO
                 N = -99999999
                 RETURN
             END IF
   20    CONTINUE
      RETURN
      END

      SUBROUTINE COMLIN(C1,CLINE,N1,N2,EFLAG)
C
C        FINDS AND INTERPRETS THE LINE NUMBERS CONTAINED ON A
C        COMMAND LINE.  CHECKS FOR ERRORS.
C
         INTEGER C1,N1,N2,I,J
         INTEGER*2 CLINE,BLNK,COMMA
         LOGICAL EFLAG
         DIMENSION CLINE(80)
         DATA BLNK/' '/,COMMA/','/
         EFLAG = .FALSE.
C        FIND FIRST DIGIT
         I = C1
         J = I
         WHILE ((CLINE(J).NE.BLNK).AND.(CLINE(J).NE.COMMA)) DO
             J = J + 1
         END WHILE
         IF (J.GE.80) THEN DO
             EFLAG = .TRUE.
         ELSE DO
C            CONVERT FIRST NUMBER TO AN INTEGER
             CALL CNVRT(CLINE,I,J-1,N1)
C            LOOK FOR SECOND NUMBER
             I = J + 1
             J = I
             WHILE (CLINE(J).NE.BLNK) DO
                 J = J + 1
             END WHILE
             IF (J .GE. 80) THEN DO
```

```
                   EFLAG = .TRUE.
              ELSE DO
                   IF (I .EQ. J) THEN DO
C                       NO SECOND NUMBER EXISTS
                        N2 = N1
                   ELSE DO
C                       CONVERT SECOND NUMBER
                        CALL CNVRT (CLINE,I,J-1,N2)
                   END IF
                   IF (N1 .GT. N2) EFLAG = .TRUE.
              END IF
         END IF
      RETURN
      END
      SUBROUTINE SEARCH(LINE,STRING,NC,MATCH,MC1)
C
C        PART OF SLED PACKAGE
C        SEARCHES 'LINE' FOR THE FIRST OCCURRENCE OF 'STRING'.
C        'MATCH' IS SET TO '.TRUE.' IF A MATCH IS FOUND.
C        'NC' IS THE NUMBER OF CHARACTERS IN 'STRING' (REQUIRED INPUT)
C        'MC1' IS AN OUTPUT INDICATING FIRST COL OF MATCH
C
         INTEGER I,J,L,NC,MC1
         INTEGER*2 LINE,STRING
         LOGICAL MATCH
         DIMENSION LINE(80),STRING(80)
C
         J = 1
         MATCH = .FALSE.
         WHILE ((.NOT.MATCH).AND.(J.LE.81-NC)) DO
              WHILE ((STRING(1).NE.LINE(J)).AND.(J.LE.81-NC)) DO
                   J = J + 1
              END WHILE
              IF (J.LE.81-NC) THEN DO
                 I = 1
                 L = J
                 WHILE ((STRING(I+1).EQ.LINE(L+1)) .AND.
     1                  ((L+1).LT.80).AND.(I+1.LE.NC)) DO
                      L = L + 1
                      I = I + 1
                 END WHILE
                 IF (I .EQ. NC) THEN DO
                      MATCH = .TRUE.
                 ELSE DO
                      J = J + 1
                 END IF
              END IF
         END WHILE
         MC1 = J
      RETURN
      END
$$
```

SLED - FORTRAN IV

```
 10/02/80  12.22.02

FILE: SLED2     FORTRAN  T1                        NAVAL POSTGRADUATE SCHOOL
C       STANDARD LINE EDITOR--FORTRAN IMPLEMENTATION
C
C       SLED VERSION FORT1.1 NPS MONTEREY 800401
C       PROGRAMMED BY:  C. F. TAYLOR, JR., CODE 55TA
C
C       FOR FURTHER INFORMATION SEE NPS TECHNICAL REPORT
C       NPS 52-80-001 BY L. A. COX, JR.
C
C
C       MAIN PROGRAM:
C
C       READS IN COMMANDS AND CALLS THE APPROPRIATE SUBROUTINE
C       IMPLEMENTS THE "EDIT" MODE
C
        COMMON /BLK1/IN,OUT /BLK5/MFLAG,ERRCT,CURLIN
        INTEGER*2 C1,C2,A,C,D,L,M,O,Q,R,S,T,V,INLINE,BLNK
        INTEGER IN,OUT,ERRCT,CURLIN,NC
        DIMENSION INLINE(30)
        LOGICAL FLAG,OPENFL,MFLAG
        DATA A/'A'/,C/'C'/,D/'D'/,L/'L'/,M/'M'/,O/'O'/,Q/'Q'/,R/'R'/, S/'S
       *'/,T/'T'/,V/'V'/,BLNK/' '/
  2000 FORMAT (' E>')
  2010 FORMAT (' -INVALID COMMAND-  ',2A1)
  2020 FORMAT (' -NO TEXT FILE OPEN-')
  2030 FORMAT (' -ONLY ONE FILE CAN BE OPENED PER SESSION IN THIS ', 'VER
       *SION OF SLED-')
C       EDIT MODE--WRITE PROMPT
        WRITE (OUT,2000)
        CALL GETLIN(INLINE,NC)
        C1 = INLINE(1)
        C2 = INLINE(2)
C       FLAG GOES FALSE AFTER 'QUIT'
        FLAG = .TRUE.
C       OPENFL GOES TRUE AFTER FILE IS OPENED
        OPENFL = .FALSE.
C       MAIN EDIT LOOP
  9000 IF (.NOT. (FLAG) ) GO TO 9001
           IF (.NOT.(.NOT.(OPENFL .OR. (C1.EQ.O) .OR. (C1.EQ.M) .OR. (C1.E
       *Q.V))) ) GO TO 9002
C               FILE NOT OPEN AND THIS IS NOT AN 'OPEN' COMMAND
C               OR A 'MENU' OR 'VERSION' REQUEST
                WRITE (OUT,2020)
                GO TO 9003
  9002     CONTINUE
C               EMULATE CASE (SWITCH) STATEMENT TO PROCESS COMMANDS
                IF (C1 .EQ. BLNK) GO TO 200
                IF (C1.NE.L) GO TO 10
                CALL LIST(INLINE)
                GO TO 200
    10          CONTINUE
                IF (C1.NE.S) GO TO 20
                CALL SCREEN(INLINE)
                GO TO 200
    20          CONTINUE
                IF ((C1.NE.R).OR.(C2.NE.S)) GO TO 30
```

```
            CALL RS(INLINE)
            GO TO 200
   30       CONTINUE
            IF ((C1.NE.R).OR.(C2.NE.L)) GO TO 40
            CALL RL(INLINE)
            GO TO 200
   40       CONTINUE
            IF ((C1.NE.A).OR.(C2.NE.L)) GO TO 50
            CALL AL(INLINE)
            GO TO 200
   50       CONTINUE
            IF ((C1.NE.D).OR.(C2.NE.S)) GO TO 60
            CALL DS(INLINE)
            GO TO 200
   60       CONTINUE
            IF (C1.NE.Q) GO TO 70
            CALL QUIT
            FLAG = .FALSE.
            GO TO 200
   70       CONTINUE
            IF (C1.NE.V) GO TO 80
            CALL VERS
            GO TO 200
   80       CONTINUE
            IF (C1.NE.M) GO TO 90
            CALL MENU
            GO TO 200
   90       CONTINUE
            IF (C1.NE.O) GO TO 100
            IF (.NOT.(OPENFL) ) GO TO 9004
               WRITE (OUT,2030)
               GO TO 9005
 9004       CONTINUE
               CALL OPEN
               OPENFL = .TRUE.
 9005       CONTINUE
            GO TO 200
  100       CONTINUE
            IF ((C1.NE.C).OR.(C2.NE.T)) GO TO 110
            CALL CT(INLINE)
            GO TO 200
  110       CONTINUE
C
C           IF PROGRAM GETS HERE, COMMAND IS INVALID
            WRITE (OUT,2010) C1,C2
            ERRCT = ERRCT + 1
C           END CASE
  200       CONTINUE
            IF (.NOT.(ERRCT.GE.2) ) GO TO 9006
               CALL MENU
               ERRCT = 0
 9006       CONTINUE
 9003    CONTINUE
         IF (.NOT.(FLAG) ) GO TO 9008
C           GET NEXT LINE
```

```
            IF (.NOT.MFLAG) WRITE (OUT,2000)
            CALL GETLIN(INLINE,NC)
            C1 = INLINE(1)
            C2 = INLINE(2)
 9008    CONTINUE
         GO TO 9000
 9001 CONTINUE
      STOP
      END

      BLOCK DATA
C
      COMMON  /BLK1/ IN,OUT /BLK3/ TFILE /BLK4/ TCHAR /BLK5/ MFLAG,ERRCT
     *,CURLIN
      INTEGER*2 TCHAR
      INTEGER IN,OUT,TFILE,ERRCT,CURLIN
      LOGICAL MFLAG
      DATA IN/5/,OUT/6/,TCHAR/'$'/,ERRCT/0/,MFLAG/.FALSE./,CURLIN/1/, TF
     *ILE/2/
      END

      SUBROUTINE LIST(CLINE)
C
C     DISPLAYS TEXT TO THE TERMINAL
C
      COMMON    /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK5/ MFLAG,ERRCT,
     *CURLIN
      INTEGER*2 BLNK,LPTR,COMMA,OUTLIN,CLINE
      INTEGER IN,OUT,I,J,N1,N2,MAXLIN,EOF,ERRCT,CURLIN,FETCH
      LOGICAL MFLAG,EFLAG
      DIMENSION CLINE(80),OUTLIN(80),LPTR(4000)
      DATA BLNK/' '/,COMMA/','/,FETCH/0/
 2100 FORMAT (' -INVALID COMMAND-')
 2110 FORMAT (' ',I4,1X,80A1)
 2120 FORMAT (' -EOF-')
C
C     IF COL 2 IS BLANK, PRINT CURLIN AND EXIT
      IF (.NOT.(CLINE(2) .EQ. BLNK) ) GO TO 9010
         CALL MEMORY (FETCH,OUTLIN,LPTR(CURLIN))
         WRITE (OUT,2110) CURLIN, OUTLIN
         GO TO 9011
 9010 CONTINUE
C        NOW CHECK FOR LINE NUMBERS IN COMMAND
         CALL COMLIN(2,CLINE,N1,N2,EFLAG)
         IF (N1 .LE. 0) N1 = 1
         IF (N1 .GT. N2) EFLAG = .TRUE.
         IF (N1 .GE. EOF) EFLAG = .TRUE.
         IF (.NOT.(EFLAG) ) GO TO 9012
            ERRCT = ERRCT + 1
            WRITE (OUT,2100)
            GO TO 9013
 9012    CONTINUE
            ERRCT = 0
```

```
                 I = N1
 9014            IF (.NOT. ((I.LE.N2).AND.(I.LT.EOF)) ) GO TO 9015
                    CALL MEMORY (FETCH,OUTLIN,LPTR(I))
                    WRITE (OUT,2110) I, OUTLIN
                    CURLIN = I
                    I = I + 1
                    GO TO 9014
 9015            CONTINUE
                 IF (I.GE.EOF) WRITE (OUT,2120)
 9013         CONTINUE
 9011 CONTINUE
      RETURN
      END

      SUBROUTINE SCREEN(CLINE)
C
C     DISPLAYS 20 LINES BEGINNING WITH CURLIN OR OTHER SPECIFIED LINE
C
      COMMON    /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK5/ MFLAG,ERRCT,
     *CURLIN
      INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,I,N1,N2,LIMIT, FETCH,N
      INTEGER*2 CLINE,LPTR,OUTLIN,BLNK
      LOGICAL MFLAG,EFLAG
      DIMENSION CLINE(80),OUTLIN(80),LPTR(4000)
      DATA BLNK/' '/,FETCH/0/
 2100 FORMAT (' -INVALID COMMAND-')
 2110 FORMAT (' ',I4,1X,80A1)
 2120 FORMAT (' -EOF-')
C
      EFLAG = .FALSE.
C     FIND OUT WHETHER USER SPECIFIED A LINE
      IF (.NOT.(CLINE(2).NE.BLNK) ) GO TO 9016
         CALL COMLIN(2,CLINE,N1,N2,EFLAG)
         IF (N1 .LE. 0) N1 = 1
         IF (N1 .GE. EOF) EFLAG = .TRUE.
         IF (.NOT. EFLAG) CURLIN = N1
 9016 CONTINUE
      IF (.NOT.(EFLAG) ) GO TO 9018
         ERRCT = ERRCT + 1
         WRITE (OUT,2100)
         GO TO 9019
 9018 CONTINUE
         ERRCT = 0
         LIMIT = MIN0(CURLIN+19,EOF-1)
         DO 10 I = CURLIN,LIMIT
         CALL MEMORY(FETCH,OUTLIN,LPTR(I))
         WRITE (OUT,2110) I,OUTLIN
   10    CONTINUE
         CURLIN = LIMIT
         IF (LIMIT .EQ. EOF-1) WRITE (OUT,2120)
 9019 CONTINUE
      RETURN
      END
      SUBROUTINE RL(CLINE)
C
C     REPLACES CURRENT LINE OR THE SPECIFIED LINE OR LINES WITH
C     ANY NUMBER OF LINES
C
      COMMON    /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK5/ MFLAG,ERRCT,
     *CURLIN
      INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,N,N1,N2,I,J,LIMIT, STORE
      INTEGER*2 CLINE,LPTR,BLNK
      LOGICAL MFLAG,EFLAG
      DIMENSION CLINE(80),LPTR(4000)
      DATA BLNK/' '/,STORE/1/
 2100 FORMAT (' -INVALID COMMAND-')
C
      N1 = CURLIN
      N2 = N1
C     DETERMINE WHICH LINE(S) TO REPLACE
      IF (.NOT.(CLINE(3).NE.BLNK) ) GO TO 9020
         CALL COMLIN(3,CLINE,N1,N2,EFLAG)
         IF (N1 .LE. 0) EFLAG = .TRUE.
         IF (N1 .GE. EOF) EFLAG = .TRUE.
         IF (.NOT. EFLAG) CURLIN = N1
 9020 CONTINUE
      IF (.NOT.(EFLAG) ) GO TO 9022
         ERRCT = ERRCT + 1
         WRITE (OUT,2100)
         GO TO 9023
 9022 CONTINUE
         ERRCT = 0
         IF (N2 .GE. EOF) N2 = EOF - 1
C        REMOVE DESIGNATED LINES
         N = N2 - N1 + 1
         DO 20 I = 1,N
         LIMIT = EOF - 2
         CALL PUSH(LPTR(N1))
         DO 10 J = N1,LIMIT
         LPTR(J) = LPTR(J+1)
   10    CONTINUE
         EOF = EOF - 1
   20    CONTINUE
C        NOW INPUT REPLACEMENT LINES
         CALL INPUT
 9023 CONTINUE
      RETURN
      END

      SUBROUTINE AL(CLINE)
C
C     INPUT TEXT AFTER LINE N
C
      COMMON    /BLK1/ IN,OUT /BLK5/ MFLAG,ERRCT,CURLIN
      INTEGER*2 BLNK,CLINE
      INTEGER IN,OUT,I,J,N,N1,N2,ERRCT,CURLIN
      LOGICAL MFLAG,EFLAG
      DIMENSION CLINE(80)
      DATA BLNK/' '/
 2100 FORMAT (' -INVALID COMMAND-')
C
C     EXTRACT LINE NUMBER FROM COMMAND LINE
      CALL COMLIN(3,CLINE,N1,N2,EFLAG)
      IF (N1 .LT. 0) EFLAG = .TRUE.
      N = N1
      IF (.NOT.(EFLAG) ) GO TO 9024
         ERRCT = ERRCT + 1
         WRITE (OUT,2100)
         GO TO 9025
 9024 CONTINUE
         ERRCT = 0
         CURLIN = N + 1
         CALL INPUT
 9025 CONTINUE
      RETURN
      END


      SUBROUTINE DS(CLINE)
C
C     PART OF SLED PACKAGE
C     DISPLAYS ALL LINES CONTAINING THE DESIGNATED STRING, POSSIBLY
C     LIMITED TO LINES N THROUGH M.
C
      COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK5/ MFLAG,ERRCT,CUR
     *LIN
      INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,N1,N2,NC,FETCH,NC1
      INTEGER*2 CLINE,LPTR,BLNK,STRING,LINE
      LOGICAL MFLAG,EFLAG,MATCH,FOUND
      DIMENSION CLINE(80),LINE(80),LPTR(4000),STRING(80)
      DATA BLNK/' '/,FETCH/0/
 2100 FORMAT (' -INVALID COMMAND-')
 2110 FORMAT (' ',I4,1X,80A1)
 2220 FORMAT (' OLD STRING?>')
 2250 FORMAT (' -NO STRING FOUND-')
C
C     DEFAULT VALUES
      FOUND = .FALSE.
      EFLAG = .FALSE.
      N1 = 1
      N2 = EOF - 1
C     DETERMINE WHETHER N1,N2 WERE SPECIFIED BY USER
      IF (.NOT.(CLINE(3) .NE. BLNK) ) GO TO 9026
         CALL COMLIN(3,CLINE,N1,N2,EFLAG)
          IF (N1 .GE. EOF) EFLAG = .TRUE.
          IF (N2 .GE. EOF) N2 = EOF - 1
          IF (N1 .LE. 0) N1 = 1
 9026 CONTINUE
      IF (.NOT.(EFLAG) ) GO TO 9028
```

```
          ERRCT = ERRCT + 1
          WRITE (OUT,2100)
          GO TO 9029
 9028 CONTINUE
          ERRCT = 0
C         FETCH STRING; ISSUE PROMPT IF NECESSARY
          IF (.NOT. MFLAG) WRITE (OUT,2220)
          CALL GETLIN (STRING,NC)
          IF (.NOT.(NC .LE. 0) ) GO TO 9030
             ERRCT = ERRCT + 1
             WRITE (OUT,2100)
             GO TO 9031
 9030     CONTINUE
             DO 20 I = N1,N2
             CALL MEMURY(FETCH,LINE,LPTR(I))
             CALL SEARCH(LINE,STRING,NC,MATCH,MC1)
             IF (.NOT.(MATCH) ) GO TO 9032
                FOUND = .TRUE.
                WRITE (OUT,2110) I,LINE
                CURLIN = I
 9032        CONTINUE
   20        CONTINUE
             IF (.NOT. FOUND) WRITE (OUT,2250)
 9031     CONTINUE
 9029 CONTINUE
      RETURN
      END
      SUBROUTINE RS(CLINE)
C
C     PART OF SLED PACKAGE
C     REPLACES THE FIRST OCCURRENCE OF STRING1 WITH STRING2 ON THE
C     CURRENT LINE OR WITHIN THE SPECIFIED RANGE OF LINES
C
      COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK5/ MFLAG,ERPCT,CUR
     *LIN
      INTEGER IN,OUT,MAXLIN,EOF,ERRCT,CURLIN,N1,N2,I,J,K,L,M, N,MC1,NC1,
     *NC2,FETCH,STORE
      INTEGER*2 CLINE,LPTR,BLNK,STR1,STR2,LINE
      DIMENSION LPTR(4000),CLINE(80),STR1(80),STR2(80),LINE(80)
      DATA BLNK/' '/,FETCH/0/,STORE/1/
      LOGICAL MFLAG,FOUND,MATCH,EFLAG
 2100 FORMAT (' -INVALID COMMAND-')
 2110 FORMAT (' ',I4,1X,80A1)
 2230 FORMAT (' OLD STRING?>')
 2240 FORMAT (' NEW STRING?>')
 2250 FORMAT (' -NO STRING FOUND-')
C
C     DEFAULT CONDITIONS
      N1 = CURLIN
      N2 = N1
      EFLAG = .FALSE.
C     INTERPRET COMMAND LINE
      IF (.NOT.(CLINE(3).NE.BLNK) ) GO TO 9034
         CALL COMLIN(3,CLINE,N1,N2,EFLAG)
         IF (N1 .LE. 0) EFLAG = .TRUE.
         IF (N1 .GE. EOF) EFLAG = .TRUE.
         IF (N2 .GE. EOF) N2 = EOF - 1
 9034 CONTINUE
      IF (.NOT.(EFLAG) ) GO TO 9036
         ERRCT = ERRCT + 1
         WRITE (OUT,2100)
         GO TO 9037
 9036 CONTINUE
         ERRCT = 0
C        READ IN TWO STRINGS: PROMPT IF NECESSARY
         IF (.NOT. MFLAG) WRITE (OUT,2230)
         CALL GETLIN(STR1,NC1)
         IF (NC1 .LE. 0) EFLAG = .TRUE.
         IF (.NOT. MFLAG) WRITE (OUT,2240)
         CALL GETLIN(STR2,NC2)
         IF (.NOT.(EFLAG) ) GO TO 9038
            ERRCT = ERRCT + 1
          . WRITE (OUT,2100)
            RETURN
 9038    CONTINUE
C        NOW FIND STRING1
         FOUND = .FALSE.
         DO 50 K = N1,N2
         CALL MEMORY(FETCH,LINE,LPTR(K))
         CALL SEARCH(LINE,STR1,NC1,MATCH,MC1)
         IF (.NOT.(MATCH) ) GO TO 9040
C           NOW MAKE SUBSTITUTION
            J = MC1
            FOUND = .TRUE.
C           DELETE STRING1
            DO 20 I = 1,NC1
            DO 10 M = J,79
            LINE(M) = LINE(M+1)
   10       CONTINUE
   20       CONTINUE
C           NOW MAKE ROOM FOR STRING2
            IF (.NOT.(NC2 .GT. 0) ) GO TO 9042
               DO 40 L = 1,NC2
               M = 81 - J
               DO 30 I = 2,M
               LINE(82-I) = LINE(81-I)
   30          CONTINUE
   40          CONTINUE
C              NOW INSERT NEW STRING
               DO 45 I = 1,NC2
               LINE(J+I-1) = STR2(I)
   45          CONTINUE
 9042       CONTINUE
C           STORE REVISED LINE
            CALL MEMORY(STORE,LINE,LPTR(K))
C           DISPLAY REVISED LINE
C           WRITE (OUT,2110) K,LINE
C           REMOVE "C" IN CC 1 OF ABOVE LINE TO ENABLE
C           DISPLAY OF EACH LINE IN WHICH A STRING HAS BEEN
C           REPLACED
```

```
              CURLIN = K
 9040     CONTINUE
   50     CONTINUE
          IF (.NOT. FOUND) WRITE (OUT,2250)
 9037 CONTINUE
      RETURN
      END
      SUBROUTINE CT
C
C     PART OF SLED PACKAGE
C     CHANGES THE MESSAGE TERMINATOR TO ANY VALID CHARACTER
C
      COMMON /BLK1/ INT,OUT /BLK4/ TCHAR /BLK5/ MFLAG,ERRCT,CURLIN
      INTEGER IN,OUT,ERRCT,CURLIN,NC
      INTEGER*2 TCHAR,INLIN,BLNK
      LOGICAL MFLAG
      DIMENSION INLIN(80)
      DATA BLNK/' '/
 2100 FORMAT (' -INVALID COMMAND-')
 2200 FORMAT (' TERMINATOR?>')
C
      IF (.NOT.(.NOT. MFLAG) ) GO TO 9044
C         ISSUE PROMPT
          WRITE (OUT,2200)
 9044 CONTINUE
      CALL GETLIN(INLIN,NC)
      IF (.NOT.((NC.EQ.0).OR.(INLIN(1).EQ.BLNK)) ) GO TO 9046
          ERRCT = ERRCT + 1
          WRITE (OUT,2100)
          GO TO 9047
 9046 CONTINUE
          TCHAR = INLIN(1)
 9047 CONTINUE
      RETURN
      END
      SUBROUTINE MENU
C
C     PART OF SLED PACKAGE
C     PROVIDES USER WITH A SUMMARY OF AVAILABLE COMMANDS
C     AND THEIR FORMATS.
C
      COMMON /BLK1/ IN,OUT
      INTEGER IN,OUT
  200 FORMAT (' SLED COMMAND SUMMARY:'//' LINE/TEXT INSERT',T38, 'STRING
     * REPLACEMENT'/3X,'ALN',T10,'INSERT <A>FTER <L>INE N', T40,'RS$P$Q$
     *',T48,'<R>EPLACE <S>TRING',/3X,'RLN',T10, '<R>EPLACE <L>INE N .OR.
     *',T40,'RSN$P$Q$',T50,'"P" WITH "Q" IN'/ 3X,'RLN,M',T15,'LINES N TH
     *RU M',T40,'RSN,M$P$Q$',T52, 'INDICATED LINES.'/' OUTPUT COMMANDS',
     *T38,'STRING SEARCH'/ 3X,'L',T10,'DISPLAY CURRENT <L>INE',T40,'DS$P
     *$',T48, '<D>ISPLAY LINES '/3X,'LN',T10,'OR LINE N.',T50, 'WITH <S>
     *TRING "P",'/3X,'LN,M',T10,'LINES N THRU M',T40, 'DSN,M$P$',T48,'OR
     * SHOW ANY LINES')
  201 FORMAT (3X,'S',T10,'<S>HOW A ', '"SCREEN" OF LINES',T50,'N-M CONTA
     *INING "P"'/3X,'SN',T10, 'SHOW A SCREEN FROM LINE N',T38,'CONTROL C
     *OMMANDS'/ 3X,'M',T10,'SHOW COMMAND <M>ENU (THIS)',T40,'Q',T48, '<Q
     *OPEN A FILE OR'/3X,'V',T10,'SHOW <V>ERSION INFORMATION', T52,'CREA
     *TE A FILE FOR EDITING'/T40,'CT  <C>HANGE THE LOGICAL'/ 5X,'TO <Q>U
     *IT THE EDIT TYPE "Q<RET>"',T40,'MESSAGE <T>ERMIN', 'ATOR')
      WRITE (OUT,200)
      WRITE (OUT,201)
      RETURN
      END
      SUBROUTINE VERS
C
C     PART OF SLED PACKAGE
C
      COMMON /BLK1/ IN,OUT
      INTEGER IN,OUT
  220 FORMAT (' SLED VERSION FORT1.1 NPS MONTEREY 800401'/ ' LOCAL EXPER
     *T IS C. TAYLOR 408-646-2691 0800-1700 PST/PDT'// ' LINE DELETE KEY
     * IS < > (ASCII) OR <CENT SIGN>  ', '(EBCDIC)'/' CHARACTER DELETE K
     *EY IS <@>'// ' EDITOR LOGICAL MESSAGE TERMINATORS ARE:'/ '     (1)
     *<RETURN>    AND     (2) <$>'// ' AND CAN BE CHANGED TO ANY STANDARD
     * FORTRAN CHARACTER.'/ ' ALL INPUT IS TRANSLATED TO UPPER CASE.'/ '
     * THE FOLLOWING DEVIATIONS FROM SLED STANDARD WERE REQUIRED:')
  221 FORMAT ( ' (1)  THE UNIVERSAL ENTRY COMMAND "SLED" INVOKES INSTRUC
     *TIONS' /'        FOR A NON-STANDARD ENTRY:  "SLED1 <FILENAME> <FILET
     *YPE> "'/      ' (2)  ONLY ONE FILE PER SESSION CAN BE OPENED.'/ ' (
     *3)  MAXIMUM FILESIZE IS 4000 LINES.'/ ' (4)  THE USER IS ASKED TO
     *INDICATE WHETHER HE IS EDITING A '/ '       NEW FILE IN ORDER TO PR
     *EVENT A DISK READ ERROR IN FORTRA N.'/' (5)  WHEN <RETURN> IS USED
     * AS A LOGICAL MESSAGE '/ '       TERMINATOR, THE LINE OR STRING IS
     *PADDED WITH BLANKS ON' /  '       THE RIGHT.  THIS AFFECTS THE RS F
     *UNCTION ONLY.')
  222 FORMAT ( ' (6)  ONLY THE FIRST OCCURRENCE OF A STRING IN EACH LINE
     *'/ '       IS REPLACED TO PERMIT FREE SUBSTITUTIONS OF BLANKS')
      WRITE (OUT,220)
      WRITE (OUT,221)
      WRITE (OUT,222)
      RETURN
      END
      SUBROUTINE OPEN
C
C     OPENS TEXT FILE AND WORKSPACE FILE
C     READS TEXT FILE INTO WORKSPACE IF IT ALREADY EXISTS
C     INITIALIZES POINTERS ETC.
C
      COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK3/ TFILE /BLK5/ MF
     *LAG,ERRCT,CURLIN /BLK6/ STACK,STKPTR
      INTEGER IN,OUT,TFILE,LINE,ERRCT,CURLIN,STORE,MAXLIN,EOF, STKPTR,I,
     *NC
      INTEGER*2 LPTR,STACK,FNAME,INLIN,YES,NO,REPLY,BLINE
      LOGICAL MFLAG
      DIMENSION LPTR(4000),BLINE(80),STACK(4000),FNAME(80),INLIN(80)
      DATA YES/'Y'/,BLINE/80*' '/,NO/'N'/,STORE/1/
 1000 FORMAT (80A1)
 2040 FORMAT (' -',I4,' LINES IN FILE:  ',80A1)
 2050 FORMAT (' -CREATING FILE:  ',80A1)
 2400 FORMAT (' FILENAME?>')
 2410 FORMAT (' IS THIS A NEW FILE?>')
 2420 FORMAT (' -MAX CAPACITY 4000 LINES EXCEEDED-')
C
C     INITIALIZE
      MAXLIN = 4000
      CURLIN = 1
C     READ IN FILENAME (COSMETIC)
      IF (.NOT. MFLAG) WRITE (OUT,2400)
      CALL GETLIN(FNAME,NC)
C     ASK WHETHER IT IS A NEW FILE (TO PREVENT FORTRAN READ ERROR)
    5 WRITE (OUT,2410)
      READ (IN,1000) INLIN
      REPLY = INLIN(1)
      IF ((REPLY.NE.YES).AND.(REPLY.NE.NO)) GO TO 5
      IF (.NOT.(REPLY.EQ.YES) ) GO TO 9048
          WRITE (OUT,2050) FNAME
          EOF = 1
          STKPTR = 1
C         ACTIVATE FILE WITH AN ACCESS
          LPTR(1) = 1
          CALL MEMORY(STORE,BLINE,LPTR(1))
          GO TO 9049
 9048 CONTINUE
C         NOW READ IN TEXT FILE
          LINE = 0
 9050     IF (.NOT. (.TRUE.) ) GO TO 9051
              READ (TFILE,1000,END=10) INLIN
              LPTR(LINE+1) = LINE + 1
              CALL MEMORY(STORE,INLIN,LPTR(LINE+1))
              LINE = LINE + 1
              GO TO 9050
 9051     CONTINUE
   10     CONTINUE
          IF (.NOT.(LINE.GE.MAXLIN) ) GO TO 9052
              WRITE (OUT,2420)
              STOP
 9052     CONTINUE
          STKPTR = LINE + 1
          EOF = STKPTR
C         TELL USER FILE OPEN
          WRITE (OUT,2040) LINE,FNAME
 9049 CONTINUE
      DO 20 I = 1,MAXLIN
      STACK(I) = I
   20 CONTINUE
      RETURN
      END
      SUBROUTINE QUIT
C
C     PART OF THE SLED PACKAGE
C     CLOSES OUT THE WORK FILE AND WRITES THE NEW OR UPDATED
C     TEXT FILE
C
      COMMON /BLK2/ LPTR,MAXLIN,EOF /BLK3/ TFILE /BLK1/ IN,OUT
      INTEGER MAXLIN,IN,OUT,EOF,TFILE,L,LIMIT
      INTEGER*2 LINE,LPTR
      DIMENSION LPTR(4000),LINE(80)
 2000 FORMAT (80A1)
 2450 FORMAT (' -',I4,' LINES WRITTEN-')
C
      REWIND TFILE
      LIMIT = EOF - 1
      DO 90 L = 1,LIMIT
      CALL MEMORY(FETCH,LINE,LPTR(L))
      WRITE (TFILE,2000) LINE
   90 CONTINUE
      WRITE (OUT,2450) LIMIT
      RETURN
      END
      SUBROUTINE MEMORY(ACTION,LINE,PTR2)
C
C     PART OF SLED PACKAGE
C     HANDLES ALL MEMORY REFERENCES USING DIRECT-ACCESS DISK FILE
C     CURRENT CAPACITY IS 4000 LINES
C     REQUIRES AT LEAST 3 DEDICATED CYLINDERS OF DISK SPACE FOR
C     WORK FILE UNDER CP/CMS ON AN IBM 360/67
C
      COMMON /BLK3/ TFILE
      INTEGER WFILE,TFILE,ACTION,STORE,PTR,AVAR,ERRS
      INTEGER*2 LINE,PTR2
      DIMENSION LINE(80)
      DATA STORE/1/
 1000 FORMAT (80A1)
C
C     DEFINE WORK FILE
      WFILE = 13
      DEFINE FILE 13(4000,80,E,AVAR)
C     CONVERT PTR2 FROM INTEGER*2 TO INTEGER
      PTR = PTR2
C     INITIALIZE READ ERROR COUNTER AND BEGIN
      ERRS = 0
      IF (.NOT.(ACTION .EQ. STORE) ) GO TO 9054
         WRITE (WFILE'PTR,1000) LINE
         GO TO 9055
 9054 CONTINUE
C        FETCH
    5    READ (WFILE'PTR,1000,ERR=99) LINE
 9055 CONTINUE
      RETURN
   99 ERRS = ERRS + 1
      IF (ERRS .LT. 10) GO TO 5
      STOP
      END
      SUBROUTINE INPUT
C
C     IMPLEMENTS THE INPUT MODE
C
      COMMON /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK5/ MFLAG,ERRCT,CUR
     *LIN
      INTEGER*2 PD,LPTR,BLNK,OUTLIN
      INTEGER MAXLIN,EOF,ERRCT,CURLIN,STORE,I,J,IN,OUT,NC
      LOGICAL MFLAG
      DIMENSION LPTR(4000),OUTLIN(80)
      DATA PD/'.'/,STORE/1/,BLNK/' '/
 2110 FORMAT (' I>')
C     IF NO INPUT IN QUEUE, PROMPT USER
      IF (.NOT.MFLAG) WRITE (OUT,2110)
      CALL GETLIN(OUTLIN,NC)
 9056 IF (.NOT. (.NOT.((OUTLIN(1).EQ.PD).AND.(OUTLIN(2).EQ.BLNK))) ) GO
     *TO 9057
         IF (.NOT.(NC .GT. 0) ) GO TO 9058
C           UNLESS IT WAS A NULL LINE
C           MAKE ROOM FOR NEW INPUT
            IF (.NOT.(CURLIN.LT.EOF) ) GO TO 9060
               J = EOF - CURLIN
               DO 10 I = 1,J
               LPTR(EOF + 1 - I) = LPTR(EOF - I)
   10          CONTINUE
               GO TO 9061
 9060       CONTINUE
               CURLIN = EOF
C              KEEPS INPUT TEXT CONTIGUOUS
 9061       CONTINUE
            EOF = EOF + 1
C           GET A NUMBER FOR NEW LINE FROM STACK
            CALL POP(LPTR(CURLIN))
C           NOW STORE THE NEW LINE
            CALL MEMORY(STORE,OUTLIN,LPTR(CURLIN))
            CURLIN = CURLIN + 1
 9058    CONTINUE
C        IF NOTHING IN QUEUE, PROMPT USER
         IF (.NOT.MFLAG) WRITE (OUT,2110)
         CALL GETLIN(OUTLIN,NC)
         GO TO 9056
 9057 CONTINUE
      RETURN
      END

      SUBROUTINE GETLIN(OUTLIN,NC)
C
C     GETS A LINE FROM TERMINAL; QUEUES UP MULTIPLE LINES
C
      COMMON    /BLK1/ IN,OUT /BLK4/ TCHAR /BLK5/ MFLAG,ERRCT,CURLIN
      INTEGER*2 INLIN,OUTLIN,QUEUE,BLNK,TCHAR
      INTEGER IN,OUT,ERRCT,CURLIN,I,J,K,LINELN,BQ,ENDQ,NC,NCHARS
      LOGICAL MFLAG,NFLAG
      DIMENSION INLIN(80),OUTLIN(80),QUEUE(80,10),NCHARS(10)
      DATA BLNK/' '/,LINELN/80/,BQ/0/,ENDQ/0/
 1010 FORMAT (80A1)
 2060 FORMAT (' -TRUNCATED; ONLY 10 ITEMS PER LINE-')
 2070 FORMAT (' -ILLEGAL CHARACTER OR BLANK COMMAND-')
C
C     MFLAG GOES TRUE WHEN MULTIPLE INPUT LINES ARE STACKED
      IF (.NOT.(.NOT.MFLAG) ) GO TO 9062
         READ (IN,1010,ERR=99,END=98) INLIN
         I = 1
 9064    IF (.NOT. ((I.LE.LINELN).AND.(INLIN(I).NE.TCHAR)) ) GO TO 9065
            OUTLIN(I) = INLIN(I)
            I = I + 1
            GO TO 9064
 9065    CONTINUE
         NC = I - 1
         IF (INLIN(I).EQ.TCHAR) MFLAG = .TRUE.
         IF (.NOT.(I.LE.LINELN) ) GO TO 9066
            DO 20 K = I,LINELN
            OUTLIN(K) = BLNK
   20       CONTINUE
 9066    CONTINUE
 9068    IF (.NOT. (INLIN(I).EQ.TCHAR) ) GO TO 9069
            IF (.NOT.(ENDQ.GE.10) ) GO TO 9070
               WRITE (OUT,2060)
               INLIN(I) = BLNK
               GO TO 9071
 9070       CONTINUE
               ENDQ = ENDQ + 1
               I = I + 1
               J = 1
               NFLAG = .TRUE.
 9072          IF (.NOT. ((I.LE.LINELN).AND.(INLIN(I).NE.TCHAR)) ) GO TO
     * 9073
                  QUEUE(J,ENDQ) = INLIN(I)
                  IF (INLIN(I).NE.BLNK) NFLAG = .FALSE.
                  I = I + 1
                  J = J + 1
                  GO TO 9072
 9073          CONTINUE
               NCHARS(ENDQ) = J - 1
               IF((I.GT.LINELN).AND.NFLAG) NCHARS(ENDQ)=0
               IF (.NOT.(J.LE.LINELN) ) GO TO 9074
                  DO 30 K = J,LINELN
                  QUEUE(K,ENDQ) = BLNK
   30             CONTINUE
 9074          CONTINUE
 9071       CONTINUE
            GO TO 9068
 9069    CONTINUE
         GO TO 9063
 9062 CONTINUE
C        GET LINE FROM QUEUE INSTEAD
         BQ = BQ + 1
         NC = NCHARS(BQ)
         DO 40 I = 1,LINELN
         OUTLIN(I) = QUEUE(I,BQ)
   40    CONTINUE
         IF (.NOT.(BQ.EQ.ENDQ) ) GO TO 9076
            BQ = 0
            ENDQ = 0
            MFLAG = .FALSE.
 9076    CONTINUE
 9063 CONTINUE
```

```
      RETURN
   98 CONTINUE
      REWIND IN
   99 CONTINUE
      WRITE (OUT,2070)
      OUTLIN(1) = BLNK
      RETURN
      END

      SUBROUTINE PUSH(X)
C
C     PUSHES A POINTER TO A FREE LINE ONTO THE STACK
C
      COMMON    /BLK1/ IN,OUT /BLK6/ STACK,STKPTR
      INTEGER STKPTR,IN,OUT
      INTEGER*2 STACK,X
      DIMENSION STACK(4000)
 2080 FORMAT (' -FREE LINE LIST STACK OVERFLOW-')
C
      IF (.NOT.(STKPTR.GT.1) ) GO TO 9078
         STKPTR = STKPTR - 1
         STACK(STKPTR) = X
         GO TO 9079
 9078 CONTINUE
C        STACK OVERFLOW
         WRITE (OUT,2080)
 9079 CONTINUE
      RETURN
      END

      SUBROUTINE POP(X)
C
C     POPS A POINTER TO A FREE LINE FROM THE STACK
C
      COMMON    /BLK1/ IN,OUT /BLK2/ LPTR,MAXLIN,EOF /BLK6/ STACK,STKPTR
      INTEGER STKPTR,MAXLIN,EOF,IN,OUT
      INTEGER*2 STACK,LPTR,X
      DIMENSION STACK(4000),LPTR(4000)
 2090 FORMAT (' -ALL SYSTEM BUFFERS FULL-')
      X = STACK(STKPTR)
      IF (.NOT.(STKPTR .LT. MAXLIN) ) GO TO 9080
         STKPTR = STKPTR + 1
         GO TO 9081
 9080 CONTINUE
         WRITE (OUT,2090)
 9081 CONTINUE
      RETURN
      END

      SUBROUTINE CNVRT(STRING,I,J,N)
C
C     CONVERTS CHARACTERS I THROUGH J OF STRING INTO AN INTEGER N
```

```
C
      INTEGER*2 STRING,DIGIT
      INTEGER I,J,N,K,L
      DIMENSION STRING(80),DIGIT(10)
      DATA DIGIT/'0','1','2','3','4','5','6','7','8','9'/
      N = 0
      DO 20 K = I,J
      L = 1
 9082 IF (.NOT. (STRING(K).NE.DIGIT(L)) ) GO TO 9083
          L = L + 1
          GO TO 9082
 9083 CONTINUE
      IF (.NOT.(L .LE. 10) ) GO TO 9084
          N = N + (L-1)*(10**(J-K))
          GO TO 9085
 9084 CONTINUE
          N = -99999999
          RETURN
 9085 CONTINUE
   20 CONTINUE
      RETURN
      END


      SUBROUTINE COMLIN(C1,CLINE,N1,N2,EFLAG)
C
C     FINDS AND INTERPRETS THE LINE NUMBERS CONTAINED ON A
C     COMMAND LINE.  CHECKS FOR ERRORS.
C
      INTEGER C1,N1,N2,I,J
      INTEGER*2 CLINE,BLNK,COMMA
      LOGICAL EFLAG
      DIMENSION CLINE(80)
      DATA BLNK/' '/,COMMA/','/
      EFLAG = .FALSE.
C     FIND FIRST DIGIT
      I = C1
      J = I
 9086 IF (.NOT. ((CLINE(J).NE.BLNK).AND.(CLINE(J).NE.COMMA)) ) GO TO 908
     *7
          J = J + 1
          GO TO 9086
 9087 CONTINUE
      IF (.NOT.(J.GE.80) ) GO TO 9088
          EFLAG = .TRUE.
          GO TO 9089
 9088 CONTINUE
C         CONVERT FIRST NUMBER TO AN INTEGER
          CALL CNVRT(CLINE,I,J-1,N1)
C         LOOK FOR SECOND NUMBER
          I = J + 1
          J = I
 9090     IF (.NOT. (CLINE(J).NE.BLNK) ) GO TO 9091
              J = J + 1
              GO TO 9090
 9091    CONTINUE
         IF (.NOT.(J .GE. 80) ) GO TO 9092
            EFLAG = .TRUE.
            GO TO 9093
 9092    CONTINUE
            IF (.NOT.(I .EQ. J) ) GO TO 9094
C              NO SECOND NUMBER EXISTS
               N2 = N1
               GO TO 9095
 9094       CONTINUE
C              CONVERT SECOND NUMBER
               CALL CNVRT (CLINE,I,J-1,N2)
 9095       CONTINUE
            IF (N1 .GT. N2) EFLAG = .TRUE.
 9093    CONTINUE
 9089 CONTINUE
      RETURN
      END
      SUBROUTINE SEARCH(LINE,STRING,NC,MATCH,MC1)
C
C     PART OF SLED PACKAGE
C     SEARCHES 'LINE' FOR THE FIRST OCCURRENCE OF 'STRING'.
C     'MATCH' IS SET TO '.TRUE.' IF A MATCH IS FOUND.
C     'NC' IS THE NUMBER OF CHARACTERS IN 'STRING' (REQUIRED INPUT)
C     'MC1' IS AN OUTPUT INDICATING FIRST COL OF MATCH
C
      INTEGER I,J,L,NC,MC1
      INTEGER*2 LINE,STRING
      LOGICAL MATCH
      DIMENSION LINE(80),STRING(80)
C
      J = 1
      MATCH = .FALSE.
 9096 IF (.NOT. ((.NOT.MATCH).AND.(J.LE.81-NC)) ) GO TO 9097
 9098    IF (.NOT. ((STRING(1).NE.LINE(J)).AND.(J.LE.81-NC)) ) GO TO 909
     *9
            J = J + 1
            GO TO 9098
 9099    CONTINUE
         IF (.NOT.(J.LE.81-NC) ) GO TO 9100
            I = 1
            L = J
 9102       IF (.NOT. ((STRING(I+1).EQ.LINE(L+1)) .AND. ((L+1).LT.80).AN
     *D.(I+1.LE.NC)) ) GO TO 9103
               L = L + 1
               I = I + 1
               GO TO 9102
 9103       CONTINUE
            IF (.NOT.(I .EQ. NC) ) GO TO 9104
               MATCH = .TRUE.
               GO TO 9105
 9104       CONTINUE
               J = J + 1
 9105       CONTINUE
 9100    CONTINUE
```

```
           GO TO 9096
 9097 CONTINUE
      MC1 = J
      RETURN
      END
```

Appendix B
SLED PASCAL Implementation
(by R. Burnham, R. Coulter, and S. Smart)
CONCEPTS

The purpose of this programming project was to implement a simple text editor to run under standard PASCAL as defined in Wirth [1972]. The specifications for the program are fairly extensive and seek to define the program in specific enough terms to ensure portability. In designing and implementing this program, the following two goals were utilized:

a. PORTABILITY. The finished program should be capable of running under any implementation of standard PASCAL.

b. STANDARDIZATION. The finished program should abide by the detailed specifications provided for the user interaction with any implementation dependent features fully documented to facilitate use by both inexperienced and experienced users.

SYSTEM DESIGN

The overall design for the text editor was heavily influenced by the strict requirements of the specification document. This specification delineated the commands that were to be implemented and their format. The primary design task involved creating an efficient system which included the required commands in an implementation independent program.

FILE MANAGEMENT. The primary purpose of any text editor is to create and modify text (character) files in an interactive manner. This problem can be separated into several functional areas. The first of these is file manipulation and management. In as much as PASCAL was designed primarily as a pedagolological aid, the language lacks extensive input/output operations. This in turn allows the various implementations to define these operations. Since this would necessarily result in operations which were not portable, it was decided to design the program to meet the requirement that the user be able to access external text files from within the program and that the program not fail if the user attempted to access a non existant file.

The Berkeley Pascal implementation for the UNIX operating system (which was used for this effort) defines three types of files, in addition to the standard input and output files. The first of these concerns explicit naming. In this case, the file name is placed in the program heading and acts as a passing "parameter" from the UNIX operating system into the program. The file so named must exist as a UNIX file.

The second type is an implicitly defined file. These files are declared in variable declaration sections of program blocks and have scope in the same manner as variables. When a block is entered, the file is created with the UNIX filename tmp.x where x is an integer representing the chronological order in which the file is used. When the block is exited, the file is destroyed.

The third type of file uses a dummy file name convention. The file name is declared in the variable declaration section but can be equivalenced to an existing file by the system functions reset and rewrite. These functions create a UNIX path between the dummy file name and the actual UNIX name, which may be supplied during execution. In the case of rewrite, if no UNIX file exists, it is created.

The dummy file name seemed ideally suited to the program requirement that the user be able to create and access files at will. However, it is necessary to know whether the file which is to be opened has been previously created. Since it is impossible for the program to access the UNIX user's directory to determine if files exist, it was decided to create a SLED directory containing the file names and sizes of all files created in the SLED environment. In the event that the user desired to edit a file which existed in his UNIX directory but not in the SLED directory, the SLED directory could be edited to include this filename. To implement this feature, it was decided to explicitly name this directory <directory> as a UNIX file. Any additional files needed as temporary storage locations to be utilized while a file was being edited could then be implemented as implicitly named files, since their existance is not important to the user (unless the amount of file space allocated to the user by the operating system is limited).

TEXT MANIPULATION. As a text editor, SLED has to carry out two basic functions. The editor must insert and delete lines of text in the file, and must search lines for pattern matching and replacement. These functions are related and are the dominant factor in the choice of an effective data structure.

PASCAL has several data structure constructors in the language which can be used to fulfil these tasks. These include the linear array, record, and pointer. The salient features of these constructors in regards to the tasks involved will now be discussed.

The PASCAL array is similar to arrays in other languages with the exception that the elements of the array may be complex data types such as records. Since the array must be statically defined, the storage can be highly efficient. Lines of text can be stored as character arrays. This has the advantage that locating a given line or character can be accomplished by simply subscripting a variable. In addition, overhead is at a minimum since no pointers, links or other devices are required. The array suffers from the disadvantage that insertions and deletions are expensive as they require copying on the average half of the array. Furthermore, since the array must be defined staticly, it is likely that much of the array will be empty at any one time.

The record data structure is similar to the array except that the elements of the record need not be of the same data type. By itself, the record offers no advantages as compared to the array. However, the record fields can be used to act as pointers or links to other records thus allowing the creation of lists or trees. Trees offer the advantage that sorting, inserting, and deleting can be carried out quite efficiently. In addition, searching is easier and more efficient than with linked lists (although not as efficiently as an array). The major disadvantage of the tree is the large amount of overhead required since each interior node must contain a link to each descendant. The linked list can be considered as a compromise of the tree structure. Insertions and deletions are still efficient, but searching requires following a string of pointers through the list. Since only one pointer is required for each node, the amount of overhead is approximately halved compared to the tree.

The pointer type represents a method of dynamic allocation of records to a linked list or tree structure. This offers the advantage that space need not be allocated until actually required. Unfortunately, the methods for allocating and de-allocating memory space via pointers is poorly defined in the language, resulting in implementation dependent designs. In particular, standard PASCAL allows pointers to records to be destroyed resulting in the creation of garbage. No garbage collection is carried out by the language to recover this memory. Therefore, a common technique is to place unused records into a "free" list. This, however, defeats the idea of dynamic allocation.

Based on these considerations, it was decided that lines within the text file would be treated as a linked list. This would facilitate the line append and line replace operations. The overhead for this structure would then be one pointer per line, which was not considered excessive. Rather than depend on the pointer type to create and manage list elements, it was decided that the list would consist of an array of line records, with each record consisting of a pointer to the next line and the contents of the line.

In representing the characters of a given line, it was necessary to decide between using another linked list for elements of a line or a character array. Again, the linked list would make character insertions and deletions efficient. Since this would require a pointer for each character, about half of the memory space allocated would be overhead. A possible compromise would be to have each list node consist of several characters. This would create difficulties in insertion and deletion and would require a complicated algorithm to implement. It was decided to represent the line as a character array. This had the benefit of making the pattern matching algorithm easier to implement as well as reducing overhead.

PARAMETERIZATION. To allow for adaptations of the program to other systems, the parameters for the data structure are defined in a constant declaration block in the main program. This allows implementations to scale the size of the data structure to the amount of memory available. To further enhance portability, it was decided to localize the input/output procedures in separate routines which could be replaced when implementing the system on other machines.

IMPLEMENTATION

The linked line list was implemented as an array of line records named <buff>. Each line record consists of an integer pointer to the array element (record) of the next line and a 120 character packed array. This size allows the creation of a line which will cover the linesize of most standard output devices. In addition, a separate record, <head>, serves as a pointer into the line list and contains the number of the first line presently in the buffer.

GENERAL STRUCTURE OF SLED. The editor basically has five categories of text processing procedures. The first of these are the control commands. There are three types of control commands: a change of logical message terminator so that the user may select the symbol or character he desires to indicate an end of a line or command, a command to exit the editor mode which will write the text file to the user's file as well as terminate the program, and a command to open a file for the user, either a new file or an existing file from his directory and close any previously opened file. As discussed in the section on design, this was implemented as a separate SLED directory. The contents of the directory consist of the UNIX file name along the total number of text lines in the file. In addition to this, a scratch file is maintained to allow updating of the directory contents.

To implement the logical terminator, it was decided to limit the terminators to printable characters. This allowed the terminator to serve as an end of line signal in the text buffer, eliminating the need for creating a separate list of line lengths or employing some other line length algorithm.

The second group of commands consists of those commands concerned with output of the text file to the CRT scree. These commands are divided into those which display a specific portion of text and those which display large blocks of text. These commands, called output commands, consist of routines to display the current line, a specific line or a designated number of lines. The user optionally selects a from-line/to-line pair of numbers for display and defaults to the current line (defined as the last line displayed or last line operated upon). Another output command screens a large block of text for the user by using a from-line input. These display commands allow the user to edit large portions of his text. Two other commands, which do not process or handle text, and are considered output commands are a procedure to display a command menu to assist the user with SLED procedures and a display of the version listing for a more sophisticated user.

The third type of commands are those which handle insertion of text into the file. The line insertion commands cause the editor to enter the insert mode (all other procedures are in the edit mode). These commands allow insertion of new lines into the text as well as replacing a specific line or a group of lines in the text. In this way, the user can create or destroy portions of his text file by linking the new lines into the buffer list.

A fourth command searches the text for a particular character or string of characters and displays them to the user. Closely related to this is the command to replace these characters and strings in the body of the text, either in a specific line, a group of lines or throughout the entire text. These two types of commands along with the line insertion commands form the basis of the text processing procedures, while the output and control commands form the basis of the text handling procedures. To implement the pattern matching routine the Knuth, Morris, Pratt algorithm (as discussed in Knuth and elsewhere) was utilized. This algorithm uses the concept of a finite automation to determine how far to advance the pattern along the target line in the event of a mismatch between the pattern and its target. This is done by creating a table of edges which represent failure in the automaton. As an example, if the pattern consists of three identical letters and the first two match but the third one does not, instead of advancing the pattern by one position relative to the target the pattern can be moved three places since the first character cannot possibly match the third character of the target. The next table determines how far to advance dthe pattern if matching fails with the ith character of the pattern. In this way, no back-tracking in the target is required and the algorithm is O(n). The diagram attached as Figure B1.1 shows the general scope of SLED as defined in the specification document. It basically shows the editor commands required by the system grouped into the five primary command areas; output, control, line insertion, string serch and string replacement. The diagram also shows the different editor command modes.

SPECIFIC STRUCTURE OF SLED. Based on the general structure of SLED as proposed by the problem specification and as diagramed above, the programming team made a detailed study of the basic routines needed, how these routines would interact and the data structures and file handling procedures that would be needed. The Figure B1.2 is a schematic of the implementation of SLED as described in this documentation. In implementing the SLED program, a top-down methodology was utilized in defining the program processes needed to meet the requirements stated above.

There are four general sections or levels to the program. The first level acts as a traffic controller for the entire program. It reads each of the user input commands and branches to the appropriate subroutine. The main program dunctions as this first level and screens the commands, eliminating the incorrect ones and processing the properly entered ones. It is the framework in which SLED performs its functions.

At this same level, the change terminator is located primarily because it does not handle the user data except to place the new terminator symbol in the text file (see procedure Changeterm documentation). It is properly classified as a special procedure in program control rather than text processing or text handling.

The second level of the program contains the bulk of the text processing procedures and consequently, the bulk of the coding. One subgroup contains the procedures which display the lines of text from the user's file. In many ways, it is similar to the output section outlined in the original evaluation. In addition to the screen line and display line procedures, we have included the string search procedure in this subgroup because it functions in a similar manner in that it is involved in the display of strings within the lines of text. This subgroup interfaces with a major subgroup of the third level, namely the commands which read the user's line number designations, translate them, and fetch the lines from the text file.

The second major subgroup of this level consists of the commands which handle the majority of the text insertions, deletions and string processing. These include appending lines, replacing lines and replacing or changing strings. These commands were grouped together due to their similarity of function and commonality of coding. They each have major subprocedures at this level and interface with the primary text handling procedures at the fourth level.

Also found at the second level are two other groups of procedures which are of less importance than the text processing procedures but are useful and necessary segments of the SLED program. The first of these are the control commands which open and close the file. They are obviously required file handling procedures and perform the functions normally expected of a text editor. They interface with key implementation dependent procedures at the third level.

The last major subgroup of the second level is the required command menu and version document. The call to these procedures is rather simple and uncomplicated. Rather than palce the documents in the PASCAL program, we decided to employ the existing UNIX directory to hold them. These files are explicitly named in the program as <menu> and <version>. The file contains the current command menu as well as the version and any changes to either of these to facilitate assisting the user can be made quickly and efficiently.

The third level of programming of SLED contains two subgroups, each of which are subroutines for major procedures in level two. The first subgroup are theroutines which transform the user line number requests into from-line/to-line pairs and fetch the appropriate lines from the user's text. They are text handling commands and are part of the output section in the original specification.

The second subgroup interacts with the open and close file routines and are key text handling procedures. They are dependent upon the implementaion of PASCAL in use on the computer system. The procedures utilize a directory file which, like the menu and version, is located in the UNIX directory.

The fourth level of SLED contains the key text hadling commands to move data in and out of the buffer when required. All of the major procedures of SLED call the read buffer and write buffer routines to move through the user's file. These procedures are also implementation dependent.

A minor procedure of the program is also found at this level which causes the command menu to be printed when the user makes two consecutive errors, a requirement of the SLED specification.

TESTING AND EVALUATION

The constraints of time prohibited an exhaustive and thorough evaluation of SLED. There has, however, been extensive and continuous testing of the modules of SLED in the initial programming phases and as the program took its finished form. While not exhaustive the testing and evaluation performed by the programming team has resulted in a fully operational and effective editor.

Once the major operating bugs were identified and removed from the program, the task of specific debugging of each comman and its interaction with the other commands of SLED was undertaken, including the testing of pathelogical errors where purposely erroneous and improper commands were input with the express purpose of causing the system to perform incorrectly or fail.

While we are satisfied that the program will function as required, there is further room for testing of the system. The limited time available precluded the testing of large files and extensive directories and the actual production of useable and functional files. Besides the testing of large files, a period of time should be spent by disinterested parties (actual users) in using SLED to produce files and testing the system. From this evaluation, any remaining system bugs should be easily identified and corrected.

SLED - PASCAL

```
{
**********************************************************************
                                                         SLED
**********************************************************************



                        Programmed by:  Robert M. Burnham

                                                        Ronald J. Coulter

                                                        Scott W. Smart

                        Naval Postgraduate School, Monterey, California

                        Specification:  20 February 1980

                        Written:  12 March 1980

                        Compiled:  14 March 1980

                        Source Computer:  PDP 11-50

                        Object Computer:  PDP 11-50

                        Language:  PASCAL

                        Implementation:  UC Berkeley PASCAL

                        File Location:  UNIX PWB /work/cs300/smart/sled.p

           Editor File Name:  sled.p
```

Documentation File Name: sled.doc

Program Access: type " SLED "

Project Specification: Issued Separately

Abstract: SLED is a very simple general purpose text editor implemented in PASCAL and designed to be relatively transportable to other PASCAL systems. It performs a minimum of the usual text editing and display features found in a typical text editor.

```
**********************************************************************

}
{*********************** BEGIN SLED ****************************************)

{

                 * DOCUMENTATION FOR THE OPERATION OF *
                 * THE BODY OF THE MAIN PROGRAM CAN   *
                 * BE FOUND AT LINE 2376              *
}


       program sled(input,menu,version,output,directory);

label 10,100;


const

    bufsize = 50;
    linesize = 120;
    nill = -1;
    namesize = 8;
    errmsg = 'INVALID COMMAND';
    blank = ' ';
        comma = ',' ;
        error = '  **  DATA ENTRY ERROR+++PLEASE REENTER DATA  ** ' ;

type

    line = record
               nextline : -1..bufsize;
               linestring : packed array[1..linesize] of char;
               end;
    buffer = array[1..bufsize] of line;
    patstring = packed array[1..linesize] of char;
    header = record
             firstline:  integer;
             ptr : -1.. bufsize;
             end;

var

       letter, c, ch, ct : char ;
       errcnt, textsize, curline : integer ;
       menu, version, mfile, tempfile, directory, tempd : text ;
    buff : buffer;
    free : -1..bufsize;
    head : header;
```

```
fname : packed array [1..namesize] of char;
fileopen : boolean;
cmderror : boolean;
```

```
{
*******************************************************************
                          PROCEDURE WRITEBUF
*******************************************************************

PURPOSE:  This procedure writes the contents of the line
          buffer to the currently open file.  mfile is
          a dummy file name.  During execution, this
          file name is replaced by a "path" to the
          UNIX file name specified during a rewrite
          or reset operation.  File tempfile is a temporary
          file used during read and write operations to
          restore the user's UNIX file.  The temp file
          exists during execution as UNIX file tmp.2.  If
          execution terminates normally, this file is removed
          from the system.

          The procedure reads from the user's file (mfile)
          to the temp file until the point where the buffer
          is to be inserted is reached.  At this point, the
          contents of the buffer are written to the temp
          file.  After the buffer lines have been transfered,
          the remaining lines of the user file are read and
          written out to the temp file.  The temp file now
          contains the complete file.  The entire temp file
          is then read and written to the user's file.

VARIABLES:  tempfile:  scratch file
            mfile:     dummy name for user's UNIX file
            fname:     actual UNIX file name
            point:     pointer to the buffer

*******************************************************************}


     procedure writebuf;

       type pointer = integer;

       var  i : integer;
            ch : char;
            point : pointer;

         begin

{
read all lines preceeding the first line of the buffer from
the user's file to the scratch file
}
```

```
        rewrite(tempfile);
        reset(mfile,fname);
        for i := 1 to (head.firstline -1) do
            begin
            while not eoln(mfile) do
               begin
               read(mfile,ch);
               write(tempfile,ch);
               end;
            end;

{
set the list pointer to the first element of the buffer
and read all lines in the list from the buffer to the scratch file
}

          point := head.ptr;
          while point <> nill do
             begin
             i := 1;
             while (i < linesize) and (buff[point].linestring[i]
                    <> ct) do
                 begin
                 write(tempfile,buff[point].linestring[i]);
                 i := i + 1
                 end;
             writeln(tempfile);
             point := buff[point].nextline;
             end;

{reset the buffer status record}

            head.ptr := nill;
            head.firstline := 0;
            free := 1;
            for i := 1 to bufsize -1 do
               buff[i].nextline := i + 1;
            buff[bufsize].nextline := nill;

{
read past the lines of the user's file which were written to the
buffer previously
}

            for i := 1 to 40 do
                   begin
                   while not eof(mfile) do
                       readln(mfile);
                   end;
```

```
{read all remaining lines from the user's file to the scratch file}

             while not eof(mfile) do
                 beoin
                 while not eoln(mfile) do
                    begin
                    read(mfile,ch);
                    write(tempfile,ch);
                    end;
                 if not eof(mfile)
                    then hegin
                    readln(mfile);
                    writeln(tempfile);
                    end;
                 end;



{write the entire scratch file into the user's file}

              rewrite(mfile,fname);
              reset(temofile);
              while not eof(tempfile) do
                  heoin
                  while not eoln(tempfile) do
                     hegin
                     read(tempfile,ch);
                     write(mfile,ch);
                     end;
                  if not eof(tempfile)
                     then heqin
                     readln(tempfile);
                     writeln(mfile);
                     end;
                  end;

       end ;           { writehuf }
```

```
{
****************************************************************
                                 PROCEDURE READBUF
****************************************************************

PURPOSE:  This procedure reads a block of 40 lines from a user
          file into the buffer.  Mfile is a dummy file name
          which is replaced during execution by a "path" to
          a UNIX file.  Strtline is the first line of the
          file to be placed in the buffer.  This line and
          the next 39 (if they exist) are read from the user's
          file and linked into the list.

VARIABLES:  mfile:  dummy filename for user's text file
            fname:  actuall UNIX file name
            strtline:  first line to be placed in buffer
            textsize:  total number of lines in user's file

*****************************************************************}


     procedure readbuf(strtline : integer);

     var ch : char;
         i,k : integer;
         numline : integer;

           begin

          {determine number of lines to be placed in buffer}

           numline := 40;
               if (textsize +1 - strtline) < numline
                   then numline := textsize +1 - strtline;
     reset(mfile,fname);
     for i := 1 to (strtline - 1) do
         readln(mfile);

{ insert lines into buffer }


        for i := 1 to numline do
                begin
                k := 1;
                while not eoln(mfile) do
                   begin
                   read(mfile,ch);
                   buff[i].linestring[k] := ch;
                   k := k + 1;
                   end;
```

```
                    buff[i].linestring[k] := c;
                    if i > 1 then huff[i-1].nextline := i;
                    readln(mfile);
                  end;
           buff[numline].nextline := nill;
           free := numline + 1;
           for k := numline + 1 to bufsize - 1 do
              buff[k].nextline := k+1;
           huff[bufsize].nextline := nill;


{update buffer status record}


             head.ptr := 1;
             head.firstline := strtline;

             end ;    { readbuf }
```

```
{
************************************************************************
                                 PROCEDURE FETCHLINE
**************************************************************************

PURPOSE:  This procedure takes the number values produced by
-------     the Lineoet procedure and causes the appropriate number
                        of lines to be printed on the CRT screen.  It is called by
                        those procedures which need to enter the users buffer
                       and extract part of the text.  It, in turn, calls the procedures
                        Readbuf and Writebuf which are part of text-buffer
                        storage system.  If the users request for text exceeds
                        the actual size of the text, this procedure will print
                        all the text that is available.

VARIABLES:  The follwing variables are used in Fetchline:
---------       x, p:  counting numbers for iterative routines
                                pointer:  number pointer to the next line
                                fline:  from-line value
                                tline:  to-line value
                       Global variables include:
                                textsize:  the number of lines in the users text
                                curline:  current line value
                                head.firstline:  first line of the users text
                                head.ptr:  pointer to the first line
                                buff[pointer].nextline:  number value of the next
                                                 line in the buffer
                                buff[pointer].linestring[p]:  character value of the
                                                 line in the buffer


**********************************************************************
}

procedure fetchline ( fline, tline : integer ) ;


var

       x,y, p, pointer : integer ;

begin


       if fline > textsize
              then

                     fline := textsize ;
       if tline > textsize
```

```
        then
               tline := textsize ;
if fline < head.firstline
         then
                begin
                        writebuf ;
                        readbuf ( fline ) ;
                end ;
 pointer := head.ptr ;
                                                         { searches for the from-line value
                                                           in the users text.  If not in the
                                                           buffer, then the rest of the file
                                                           is searched. }

 for x := head.firstline to ( fline - 1 ) do
 begin
                  pointer := buff[pointer].nextline ;
                  if pointer = nil)
                          then
                                  begin
                                           writebuf ;
                                           readbuf ( x ) ;
                                           pointer := head.ptr
                                   end ;
  end;
                                                   { print out each line, character by
                                                     character, from the buffer to the
                                                     CRT for the from-line/to-line values.
                                                     If the buffer is exceeded, then the
                                                     next block of text is read to the
                                                     buffer. }

  for x := fline to tline do
                                                                   begin
              y := x;
                           write ( '    ',y: 5,'    ' ) ;
            p := 1;
                            while (buff[pointer].linestring
                and (p < linesize) do
                                begin

                                        write( buff[pointer].linestring[p] ) ;
                                        p := p + 1

                                end ;
                       writeln ;
       if x < tline then begin
                       pointer := buff[pointer].nextline ;
                       if pointer = nill
                               then

                                       begin
                                               writebuf ;
                                               readbuf ( y ) ;
                                               pointer := head.ptr
                                       end ;
          end;
              end ;

                                                             { reset the value of the current line }
   curline := tline ;

end:  {fetchline}
```

```
{
********************************************************************
                                     PROCEDURF LINEGET
********************************************************************

PURPOSE:  This procedure, called by some of the main procedures of
-------    SLED, reads the line number of the user input instructions,
                 constructs a " from-line/to-line " pair of variables and
                     checks for errors in user input.  The user can designate the
                     current line with a carriage return or a change terminator
                     command, insert a value for any other line desired or
                     specify lines from one number to another.
                           The values produced by this procedure are passed back to
                     the calling command.  This procedure is called by Displayline
                     and Screenline.

VARIABLES:  The following local variables are used:
---------                 line:  the from-line value
                                 toline:  the to-line value
                                  screencheck:  boolean check value to detect an error
                                          and signal the calling procedure to take specific
                                          action
                                  num:  value of the first number read ( to-line value )
                                  tnum:  value of the second number read ( from-line )
                                  temp:  tempory variable for numbers
                                  check:  boolean checker for an end-of-line character
                          Global variable reference:
                                   curline:  current line
                                   ct:  change terminator character
                                   errcnt:  the current user error count for the
                                                    Erroroutine procedure

********************************************************************
}


procedure lineget ( var line,toline : integer; var screencheck : boolean ) ;


var

       num, tnum, temp : integer ;
       check : boolean ;



begin
```

```
                                                 { Reads in the first number and
                                                   converts it to the from-line value }
    check := false ;
    screencheck := true ;
    temp := 0 ;
    line := curline ;
    toline := line ;


begin
   if not eoln(input) then begin
     read (c) ;
     if c in ['0'..'9']
           then


                  begin
                  repeat
                  if eoln (input)
                        then

                              check := true ;
                  temp := 10 * temp + ord (c) - ord ('0') ;
                  num := temp ;
     if not eoln(input) then
                  read (c) ;
                  until not ( c in ['0'..'9'])  or ( check ) or  ( c = ct )
                  end ;

         if ( check ) or   ( c = ct )
               then
               begin

                     line := num ;
                     toline := line ;
               end ;
               temp := 0 ;

     end;
                                                               { Reads in the second number
                                                                 and converts it to the :o-line
                                                                 digit }

     if not (check)  and ( c = comma )
           then

                  begin
                    if c = comma
                         then
```

```
                                     then
                             read (c) ;
                             until ( eoln(input) ) or ( c = ct )
                      end ;
      end;
       if line > toline
               then
                 begin
                      writeln(error) ;
                      writeln('      Data is entered " fromline, toline" ') ;
                      line := curline ;
                      errcnt := errcnt + 1 ;
                      toline := line ;
                 end ;
       errcnt := 0 ;
  if line = 0
       then
              line := curline ;

end ;    { lineget }
```

```
{
******************************************************************
                                          PROCEDURE DISPLAYLINE
******************************************************************

PURPOSE:  This procedure diplays lines of text from the users
-------      file.  The user requests the current line, a specific
                          line or a group of lines with a starting value and
                          an ending value.  The input command is < L value(,value) >.
                                 The number commands are read by calling Lineget and
                          the actual lines are fetched and printed by Fetchline.
                          This procedure is merely a vehicle for the interaction
                          of these two procedures.

VARIABLES:  The following variables are used:
---------        linefrom:  from-line value
                                 lineto:  to-line value
                                 check:  error detecting variable

******************************************************************
}

procedure displayline ;

var

        linefrom, lineto : integer ;
        check : boolean ;

begin

        lineget ( linefrom, lineto, check ) ;
    if linefrom = 0 then linefrom := 1;
       if textsize <> 0 then
         fetchline ( linefrom, lineto )
         else writeln('0 lines in file');
      if eoln(input) then begin
       readln;
        write('E>');
        end;

end ;   { displayline }
```

```
{****************************************************************
                                          PROCEDURF SCREENLINE
****************************************************************

PURPOSE:  This procedure displays 20 lines of text to the user.  It
                    can be started at the current line or at any line
                    designated by the user.  If the request exceeds the text
                    size, then the screen will terminate with the last line
                    line of the text.  The input command for displayline is
                    < S value >.  If improper data values are input, the checker
                    will print only the current line with the error diagnostic
                    printed by the Lineget procedure.
                            The procedure calls the procedures Lineget and
                    Fetchline.


VARIABLES:  The following local variables are used:
---------         linefrom:  from-line value
                                 lineto:  to-line value
                                 checker:  boolean value passed by Lineget which
                                           causes only the current line to print instead
                                           of 20 lines.  Done on error only
                            Global variable used:
                                 curline:  current line

*******************************************************************
}


procedure screenline ;


var

        linefrom, lineto : integer ;
        checker : boolean ;

begin

        lineget ( linefrom, lineto, checker ) ;
        if linefrom = 0
                then
                        linefrom := linefrom + 1 ;
        if linefrom <= lineto
                then
                lineto := linefrom + 20 ;
                                                                                              { Error routine for invalid
                                                                                                data in Lineget procedure }
        if checker = false
                then
```

```
                        begin
                                linefrom := curline ;
                                lineto := linefrom
                        end ;
    if textsize <> 0 then
        fetchline ( linefrom, lineto )
     else writeln('0 lines in file');
          if eoln(input) then begin
              readln;
             write('E>');
              end;

end ;   { screenline }
```

```
{
****************************************************************
                                     PROCEDURE APPENDLINE
*****************************************************************

PURPOSE:  This procedure inserts lines into the buffer.
          The parameter strtline is the line which the
          inserted line(s) is to follow.  If this line
          is not currently in the buffer, the contents
          of the buffer are written out and the buffer
          is refilled begining with the line strtline.
          The procedured then continues to insert lines
          until the end of input symbol is reached.
          If the buffer is filled during this operation
          the contents of the buffer are written out
          and the last line inserted becomes the first
          (only) line in the buffer.

VARIABLES:
            point:  pointer to the line in the buffer the
                      inserted lines are to follow
            epoint:  pointer to the line immediately following
                     the lines inserted
            bufline:  counter to keep track of the line number
                      which point is pointing to

***********************************************************************}

     procedure appendline (var strtline : integer);

         type pointer = -1..bufsize;

         var text : packed array [1..linesize] of char;
             point,epoint : pointer;
             done : boolean;
             i : 1..linesize;
             bufline : integer;
             inpch : char;

begin

{test if inserted lines are to be placed before the first line
of text.  If so, pointer is nill.  If the file is empty the
first inserted line becomes the first line of text and the
buffer status is updated }
     if strtline = 0
         then begin
               if head.firstline > 1
                     then begin
                          writebuf;
                          readbuf(1);
                     end;
               if head.firstline = 0
                        then begin head.ptr := nill;
                        head.firstline := 1;
                        end;
               epoint := head.ptr;
               point := nill;
         end

{find the start line in the buffer}

         else begin
             if head.firstline > strtline
                then begin
                    writebuf;
                    readbuf(strtline);
                    point := head.ptr;
                end
                    else begin
                    point := head.ptr;
                    bufline := head.firstline;
                    while bufline < strtline do
                        begin
                        point := buff[point].nextline;
                 {if start line not in file, get start line}
                        if point = nill
                           then begin
                            writebuf;
                            readbuf(strtline);
                            point := head.ptr;
                            bufline := strtline;
                            end
                            else begin
                            bufline := bufline + 1;
                            end;
                        end;
                 end;

           {after the start line is found, set epoint to
            the next line}
                 if buff[point].nextline <> nill then
                    epoint := buff[point].nextline
                    else epoint := nill;


         end;
           done := false;
```

```
{read inserted text lines until the line contains
the end of input symbol (.).                    }

              while not done do
                  begin
                  i := 1;
                  if not eoln(input) then
                      read(inpch)
                      else inpch := ' ';
                  text[i] := inpch;
                  if inpch = '.'          {check for end of input}
                       then begin
                          if not eoln(input)
                             then begin
                             read(inpch);
                             i := i + 1;
                             text[i] := inpch;
                             end;



                            {end of input. link epoint line to
                              last inserted line}

                           if (i = 1) or (inpch = ct)
                               then begin
                                    if point <> nil
                                      then begin
                                         buff[point].nextline := epoint;
                                      end;
                                    done := true;
                               end;
                           end;

               {write text line into buffer}


                    if not done
                        then begin
                        textsize := textsize + 1;
                          while not eoln(input) and (inpch <> ct)
                                      and(i < linesize) do
                               begin
                               i := i + 1;
                               read(inpch);
                               text[i] := inpch;
                               end;
                          if eoln (input)
                               then begin
                        readln;
                        write('[>');
                        text[i + 1] := ct;
                        end;
           {if no free lines available, write out
                contents of buffer}


                if free = nill
                     then begin
                     if point <> nill then
                         buff[point].nextline := epoint;
                     writebuf;
                     readbuf(strtline);
                     point := buff[head.ptr].nextline;
                     epoint := buff[point].nextline;
                     end;
                if point <> nill
                     then buff[point].nextline := free
                     else head.ptr := free;
                point := free;
                strtline := strtline + 1;
                free := buff[free].nextline;
                buff[point].linestring := text;
                {update current line}
                curline := curline + 1;
            end;
     end;

end ;  { appendline }
```

```
{
****************************************************************
                                   PROCEDURE REPLACELINE
****************************************************************

PURPOSE:  This procedure serves two purposes.  First it deletes
          all lines between the parameters strtline and
          endline (inclusive).  Next procedure calls
          procedure appendline allowing the user to add
          any lines in place of the deleted ones.
          If the lines to be deleted are not currently in
          the buffer, the buffer is written out and the
          first line to be deleted becomes the first line of
          the buffer.

VARIABLES:

            point:  points to the first line to be deleted
            epoint:  points to the line after the last line to
                     be deleted
            bufline:  the line to which point is pointing

****************************************************************}


     procedure replaceline(var strtline, endline : integer);

           type pointer = -1..bufsize;


                var point,epoint : pointer;
                bufline : integer;
                temp : integer;

     begin
    {find startline in buffer.  If not in buffer write contents of
buffer to output file and input 40 lines beginning with startline.}

if head.firstline > strtline
      then begin
           writebuf;
           readbuf(strtline);
           point := head.ptr;
           end
      else begin
           point := head.ptr;
           bufline := head.firstline;


      while bufline < strtline -1 do
```

```
           begin
           point := buff[point].nextline;
           if bufline < textsize then
              if point = nill then
                  begin
                  writebuf;
                  readbuf(strtline);
                  point := head.ptr;
                  bufline := strtline;
                  end
                  else begin
                  bufline := bufline + 1;
                  end;
        end;
     'end;
{
determine which lines are to be replace and link these lines to the
free list.  If no new lines are to be added, reconnect the lines
in the buffer without the deleted lines.  (i.e. connect strtline - 1
to endline).  If new lines are to be added, replace the startline
with the first new line and append any following lines to it, then
reconnect the following lines beginning with endline.
}

     epoint := point;                {find the endline}
     for bufline := strtline -1 to (endline -1) do
           begin
           epoint := buff[epoint].nextline;
            if bufline < textsize then
            if epoint = nill
                 then begin
                 buff[point].nextline := nill;
                 writebuf;
                 readbuf(bufline);
                 point := 0;
                 epoint := head.ptr;
                 end;
          end;

{update the textsize and buffer status record}

     textsize := textsize - (endline - strtline ) -1;
     temp := buff[point].nextline;
     buff[point].nextline := buff[epoint].nextline;
     buff[epoint].nextline := free;
     free := temp;
     strtline := strtline -1;
     appendline(strtline)

     end ;    { replaceline }
```

```
{
*****************************************************************
                                            PROCEDURE STRINGDISP
*******************************************************************

PURPOSE:  This procedure searches the text between the parameters
           strtline and endline for any lines containing an
           occurence of the string <pattern>.  If the string
           is found, the line containing it is displayed by
           calling procedure fetchline.
           The procedure first computes the next table to
           implement the Knuth, Morris, Pratt string search
           algorithm.  This table is then used in determining
           how far to move the pattern along the line of text
           in the event of a non-match.


VARIABLES:
          next : next table
          point:  pointer to the textline currently being
                  evaluated.
          patlength:  the length of the pattern

*********************************************************************}


     procedure stringdisp(var strtline,endline : integer; var
          pattern:  patstring; var patlength:integer);

          type pointer = -1..bufsize;

                  var

                           i, j, bufline : integer ;
              next :  array [1..linesize] of integer;
              point : pointer;
              text : packed array[1..linesize] of char;
                     done : boolean;



     begin
{compute next table for string matching procedure }

     i := 0;
     next [1] := 0:
     j := 1;
     while j < patlength do
       begin
       done := false;
```

```
          repeat
            if i > 0 then
            if pattern[i] <> pattern[j]
                then i := next[i]
                else done := true;
          until (i <= 0) or done;
          i := i + 1;
          j := j + 1;
          if pattern[i] = pattern [j]
              then next[j] := next[i]
              else next[j] := i;
          end;


                                                                    { find strtline in the buffer }

  if head.firstline > strtline
         then begin
              writebuf;
              readbuf(strtline);
              point := head.ptr;
         end
         else begin
              point := head.ptr;
              bufline := head.firstline;
              while bufline <strtline do
                   begin
                   point := buff[point].nextline;

                 {if line is not in buffer, write out buffer
                  and read in current line}

                 if point = nill
                      then begin
                             writebuf;
                             readbuf(strtline);
                             point := head.ptr;
                             bufline := strtline;

                      end
                      else begin
                             bufline := bufline + 1;
                      end;
                 end;
  end;
  while bufline <= endline do
       begin
       text := buf [point].linestring;
       i := 1;
       j := 1;
       {begin pattern matching}

        while (i <= patlength) and(j <= linesize)
           and (text[j] <> ct) do
         begin
         if pattern[i] = text[j]
    then begin
        i := i + 1;
        j := j + 1;                                {keep matching}
    end
                                                 {match not found
                                                  advance pattern
                                                  as determined by
                                                  next table}
     else begin
        if next[i] > 0
            then i := next[i]
            else begin
                 i := 1;
                 j := j + 1;
            end;
     end;
  end;

                           {pattern matches}
  if i> patlength then fetchline(bufline,bufline);
  point := buff[point].nextline;
  bufline := bufline + 1;
  if bufline < endline then
     if point = nill
              then begin
                  writebuf;
                  readbuf(bufline);
                  point := head.ptr;
              end;
     end;
end ;  { stringdisp }
```

```
{
******************************************************************
                                         PROCEDURE STRINGREPL
******************************************************************

PURPOSE:  This procedure searches the user's text file between
          the parameters strtline and endline for any occurances
          of the string <pattern>.  If <pattern> is found in a
          line, it is replaced by the string <string> and the
          resulting line is displayed.  The procedure uses
          the Knuth, Morris, Pratt algorithm to match the
          pattern in the line.  The next table is first
          computed.  This table is then used in computing
          how far to move the pattern along the text line
          in event of a non-match.

VARIABLES:  patlength:  length of the string <pattern>
            strlength:  length of the new string <string>
            text:       a line buffer for the current line
            next:       next table
            point:      pointer to the current line

******************************************************************}

    procedure stringrepl (var strtline, endline : integer;
              var pattern, string : patstring;
              var patlength, strlength : integer);


  type pointer = -1..bufsize;

      var
       i,j,k,m,temp,temp2 : integer;
       bufline : integer;
       next : array[1..linesize] of integer;
       point : pointer;
       text : packed array[1..linesize] of char;
       found : boolean;
       done : boolean;

begin
                                                                 { compute next table }
     i := 0;
     j := 1;
     next[1] := 0;
     while j < patlength do
         begin
            done := false;
            repeat
               if i > 0 then
              if pattern [i] <> pattern[j] then i := next[i]
                    else done := true;
    until (i <= 0) or done;
i := i + 1;
j := j + 1;
if pattern[i] = pattern[j]
      then next[j] := next[i]
      else next[j] := i;
 end;

                                                           { find start line in buffer }

if head.firstline > strtline
     then begin
     writebuf;
     readbuf(strtline);
     point := head.ptr;
     end
     else begin
     point := head.firstline;
     while bufline < strtline do
       begin
        point := buff[point].nextline;
        if point = nil)
           then begin
           writebuf;
           readbuf(strtline);
           point := head.ptr;
           bufline := strtline;
           end
           else begin
           bufline := bufline + 1;
           end
        end;
      end;
                                         { try to match pattern in text line }

while bufline <= endline do
   begin
   found := false;
   text := buff[point].linestring;
   i := 1;
   j := 1;
   temp := 1;
   temp2 := 1;
    while (i <= patlength) and (j <= linesize)
```

```
        and (text[j] <> ct) do
    beaom
  if pattern[i] = text[j]
       then begin
        i := i + 1;
        j := j + 1;                  {keep matching}
        end
        else begin              {no match}
           if next[i] > 0
              then i := next[i]
              else begin
              i := 1;
              j := j + 1;
              end;
         end;

           {pattern match found-replace pattern with
            string}


     if i > patlength

        then begin
            found := true;
            for k := temp2 to (j - (patlength + 1)) do
                begin
                buff[point].linestring[temp] := text[k];
                temp := temp + 1;
                end;
            m := 1;
            for k := (j - patlength) to ((j - patlength)+
                        (strlength - 1)) do
                begin
                if temp < linesize then
                   buff[point].linestring[temp] := string[m];
                    temp := temp + 1;
                    end;
                temp2 := j;
                i := 1;
             end;         {continue to search for pattern}
       end;
        if found then begin
              m := temp2;
              if temp < linesize then
                 for k := temp to (temp + (j - temp2)) do
                     begin
                        if k < linesize then
                          buff[point].linestring[k] := text[m];
                     m := m + 1;
                     end;
                 {display new line }

             fetchline(bufline,bufline);
          end;
    point := buff[point].nextline;
    bufline := bufline + 1;
    if bufline < endline
      then if point = nill
            then begin
            writebuf;
            readbuf(bufline);
            point := head.ptr;
            end;
    end;

end;    {stringrepl}
```

```
{
**********************************************************
           PROCEDURE APPENDCOM
**********************************************************

PURPOSE:  This procedure computes the start line for
          the insert procedure (appendline).  If the
          line number is defaulted, the current line is
          used.  If the line number exceeds the number of
          lines in the text, or is otherwise invalid, an
          appropriate error message is returned.

VARIABLES:
           lineno : start line for text insertion
           inv :    true if invalid command

*************************************************************}

     procedure appendcom;

     var com: char;
         lineno : integer;
         inv : boolean;

     begin

     lineno := 0;
     com := ' ';
     inv := false;
     if eoln(input) or (com = ct) then if textsize > 0
         then lineno := curline;
     while not eoln(input) and (com <> ct) do
        begin
        read(com);
        if not (com in (['0'..'9'] + [ct]))
          then begin         {compute line number}
          if not eoln(input) then repeat read(com) until eoln(input);
          writeln(errmsg);
          errcnt := errcnt + 1;
          inv := true;
          end
          else if com <> ct then
             lineno := (10 * lineno) +(ord(com) - ord('0'));
        end;
     if not inv then
        if (textsize < lineno)
             then begin
             writeln(textsize:1,' lines in file');
             if not eoln(input) then repeat read(com) until eoln(input);
             end
        else begin
        if eoln(input)
              then begin
              readln;
              write('I>');
              end;
        appendline(lineno);
        errcnt := 0;
        end;


if eoln(input)
   then begin
   readln;
   write('E>');
   end;
curline := lineno;
end;                 {appendcom}
```

```
{
**********************************************************
              PROCEDURE REPLCOM
**********************************************************

PURPOSE:  This procedure computes the start line and
          end line for the line delete procedure (replaceline).
          If the line number is defaulted, the current line
          is deleted.  If the end line is not specified, the
          startline is deleted.  If an invalid line number is
          inputed, and error message is returned.  If the
          start line or end line is greater than the text size
          the last text line is deleted.

VARIABLES:
           linest :  start line for deletion
           linend :  end line for deletion
           inv    :  true if invalid line number is inputed

*************************************************************}

     procedure replcom;

          var
        com: char;
        first : boolean;
        linest, linend : integer;
        inv : boolean;

     begin

     inv := false;
     linest := 0;
     linend := 0;
     first := true;
     com := ' ';
     while not eoln(input) and (com <> ct) do
          begin
          read(com);
          if not (com in (['0'..'9'] + [',',ct]))
              then begin
              if not eoln(input) then repeat read(com) until eoln(input);
              writeln(errmsg);
              errcnt := errcnt + 1;
              inv := true;
              end
              else if com <> ct
                    then if first     {compute start line}
                       then if com = ','
                             then first := false
                       else linest := (10*linest) + (ord(com) -
                            ord('0'))
                   else if com = ','
                       then begin
                       if not eoln(input) then repeat
                          read(com) until eoln(input);
                       writeln(errmsg);
                       errcnt := errcnt + 1;
                       inv := true;
                       end              {compute end line }
                       else linend := (10*linend) + (ord(com)
                          - ord('0'));
     end;
     if not inv then begin
         if linest = 0 then linest := curline;   {start line becomes
                                                    current line}
         if first = true then linend := linest;
         if (textsize < linest) or (textsize < linend)
                or (linend < linest)
             then begin
             writeln(textsize:1,' lines in file');
             end
              else begin
                 if eoln(input) and (textsize > 0)
                    then begin
                    readln;
                    write('I>');
                    end;
           if textsize > 0 then
              replaceline(linest,linend)
              else writeln('0 lines in file');
           end;
     end;
if eoln(input)
    then begin
    readln;
    write('F>');
    end;
curline := linest;

end ;   { replcom }
```

```
{
****************************************************************
                                                PROCEDURE DISPCOM
*******************************************************************

PURPOSE:  This procedure computes the start line, end line
          and pattern for the string search procedure (stringdisp)
           If the line number is defaulted, all lines in the
           user's file are searched.  If the pattern string is
           defaulted, the procedure prompts the user.  If an
           invalid line number is entered, an error message is
           returned.  The procedure also computes the length
           of the pattern string.
VARIABLES:
          linest : start line
          linend : end line
          string :  pattern to be matched
          stsize : length of the pattern
****************************************************************}

     procedure dispcom;

        var
          com : char;
          linest, linend, stsize : integer;
          string : patstring;
          first, stg  : boolean;
          inv : boolean;

     begin

     linest := 0;
     inv := false;
     linend := 0;
     com := ' ';
     stsize := 0;
     first := true;
     stg := false;
     while not eoln (input) and (com <> ct) do
          begin
          read(com);
          if not stg
             then if com = ct
                    then begin
                    stg := true;
                    com := ' ';
                    end
                    else if not (com in (['0'..'9'] + [','])) 
                          then begin
                          if not eoln(input) then repeat read(com)
```

```
                       until eoln(input);
                       writeln(errmsg);
                       errcnt := errcnt + 1;
                       inv := true;
                       end
                       else if first       {compute start line }
                         then if com = ',' then first := false
                         else linest := (10*linest) +(ord(com)-ord('0'))
                        else if com = ','
                              then begin
                              if not eoln(input) then repeat read(com)
                                  until eoln(input);
                              writeln(errmsg);
                              errcnt := errcnt + 1;
                              inv := true;
                              end       {compute end line}
                              else linend := (10*linend) +
                               (ord(com)-ord('0'))
                 else if com <> ct
                       then     {compute pattern string and string length} begin
                       stsize := stsize + 1;
                       string[stsize] := com;
                       end;
     end;
          if not inv then begin
                if stsize = 0
                   then    {if no string prompt user} begin
                   write('string?>');
                   com := ' ';
                   if eoln(input) then readln;
                   while not eoln(input) and (com <> ct) do
                     begin
                     read(com);
                     if com <> ct
                        then begin
                        stsize := stsize + 1;
                        string[stsize] := com;
                        end;
                    end;
                end;

{check for line defaults and set to current line.  If line is
greater than text size, set to last line of text}

          if linest = 0
                then begin
                linest := 1;
                linend := textsize;
                end
                else if first
```

```
               then linend := linest;
     if linend > textsize then linend := textsize;
     if linest > textsize then linest := textsize;
     if (linend < linest)
           then begin
           if not eoln(input) then repeat read(com)
               until eoln(input);
           writeln(errmsg);
           errcnt := errcnt + 1;
           end
           else if textsize > 0 then begin
              stringdisp(linest,linend,string,stsize);
              errcnt := 0;
              end
              else writeln('0 lines in file');
     end;
if eoln(input) then begin
     readln;
     write('E>');
     end;
curline := linend;

end ;                 { dispcom }
```

```
{
**********************************************************************
                                                  PROCEDURE STREPCOM
**********************************************************************

PURPOSE:  This procedure computes the start line, end line,
          pattern string, and new string to be inserted
          for the string replace procedure (stringrepl).
          If the line number is defaulted, the current line
          is used.  If no end line is specified, the start
          line is used.  If the new string or pattern string
          is defaulted, the user is prompted.  If an invalid
          line number is entered, an error message is returned.

VARIABLES:
           linest : startline
           linend : endline
          oldstring : pattern string to be replaced
          newstring : new string to be inserted
          oldstsize : pattern string length
          newstsize : new string length

**********************************************************************}


    procedure strepcom;

       var
         com : char;
         linest, linend, oldstsize, newstsize : integer;
         oldstring : patstring;
         first, oldstg, newstg  : boolean;
         newstring : patstring;
         inv : boolean;

    begin

    inv := false;
    linest := 0;
    linend := 0;
    com := ' ';
    oldstsize := 0;
    newstsize := 0;
    newstg := false;
    first := true;
    oldstg := false;

    while not eoln (input) and (com <> ct) do
         begin
         read(com);
```

```
     if not oldstg
        then if com = ct    {compute start, end line}
            then begin
            oldstg := true;
            com := ' ';
            end
            else if not (com in (['0'..'9'] + [',']))
                then begin
                if not eoln(input) then repeat read(com)
                    until eoln(input);
                writeln(errmsg);
                errcnt := errcnt + 1;
                inv := true;
                end
                else if first     {compute start line}
                   then if com = ',' then first := false
                       else linest := (10*linest)+(ord(com)-ord('0'))
                       else if com = ','
                            then begin
                            if not eoln(input) then repeat read(com)
                               until eoln(input);
                            writeln(errmsg);
                            errcnt := errcnt + 1;
                            inv := true;
                            end       {compute end line }
                            else linend := (10*linend) +
                             (ord(com)-ord('0'))
            else if not newstg     {input pattern string}
            then if com <> ct
                  then begin
                  oldstsize := oldstsize + 1;
                  oldstring[oldstsize] := com;
                  end
                  else begin
                  newstg := true;
                  com := ' ';
                  end
               else if com <> ct       {input new string}
                  then begin
                  newstsize := newstsize + 1;
                  newstring[newstsize] := com;
                  end;

end;

      if not inv then begin
         if oldstsize = 0     {if no pattern prompt user}
           then begin
           if eoln(input) then readln;
           write('oldstring?>');
```

```
                com := ' ';
                while not eoln(input) and (com <> ct) do
                  begin
                  read(com);
                  if com <> ct
                    then begin
                     oldstsize := oldstsize + 1;
                     oldstring[oldstsize] := com;
                     end;
                  end;
               end;

               if newstsize = 0       {if no new string prompt user}
                 then begin
                 if eoln(input) then readln;
                 write('newstring?>');
                 com := ' ';
                     while not eoln(input) and (com <> ct) do
                       begin
                       read(com);
                       if com <> ct
                         then begin
                         newstsize := newstsize + 1;
                         newstring[newstsize] := com;
                         end;
                      end;
                   end;

{check for line defaults and set default values.  If line number
is larger than text size, set line number to the last line
of text.}

          if linest = 0 then linest := curline;
          if first
               then linend := linest;
          if linest > textsize then linest := textsize;
          if linend > textsize then linend := textsize;
          if (linend < linest)
                then begin
                if not eoln(input) then repeat read(com)
                until eoln(input);
                writeln(errmsg);
                errcnt := errcnt + 1;
                end

                 else if textsize > 0 then begin
                     stringrepl(linest,linend,oldstring,
                        newstring,oldstsize,newstsize);
                     errcnt := 0;
                     end
                    else writeln('0 lines in file');
     end;

curline := linend;
if eoln(input) then begin
     readln;
     write('E>');
     end;

end ;       { strepcom }
```

AD-A116 757   NAVAL POSTGRADUATE SCHOOL   MONTEREY CA   F/G 5/8
BASELINE IMPLEMENTATIONS OF THE STANDARD LINE EDITOR (SLED).(U)
OCT 80   L A COX, R J COULTER, C F TAYLOR
UNCLASSIFIED   NPS52-80-012   NL

2 of 2

END
DATE
FILMED
8 82
DTIC

```
{
*************************************************************************
                                                    PROCEDURE GETFILE
*************************************************************************

PURPOSE:  This procedure inputs the user's file name.  The SLED
          directory is then searched for that file name.  If
          the file name is found, the file is opened and textsize
          is set to the number of lines in the file.  If the
          file name is not found, a new file is created and the
          file name is entered in the directory.  The file
          <directory> is an explicit UNIX file which contains the
          current SLED directory.  The file <tempd> is a scratch
          file which is created during execution as UNIX file
          tmp.1.  The contents of the directory are read to the
          scratch file during the directory search (less the file
          to be opened, if found).  This scratch file is then
          read to the directory with the updated number of text
          lines for the currently opened file in procedure close.

VARIABLES:
           directory :  SLED directory file.  The format for the file
                        is <text size> <file name>.
           tempd :      scratch file
           fname :      user's UNIX file name to be opened
           textsize :   number of lines in the file

*************************************************************************}




     procedure getfile;

     var       ch : char;
               found : boolean;
                               index, j, k, i : integer ;

     begin

     ch := ' ';
     textsize := 0;
     index := 0;

{input the user file name}

     for i := 1 to namesize do
         fname[i] := ' ';
     i := 0;
     while not eoln(input) and (ch <> ct) do
          begin
              repeat read(ch) until (ch <> ' ') or eoln(input);
           if ch = ct
                 then
                 i := namesize + 1
                 else begin
                 i := i + 1;
                 if i <= namesize then fname[i] := ch;
                 end;
           end;

{search for the user file fname in the directory}

     if fname <> '          ' then begin
          reset(directory);
          found := false;
          i := 1;
          while not eof(directory) and not found do
              begin
              textsize := 0;

{if the file is found, compute textsize}

              repeat
                 read(directory,ch);
                 if ch in ['0'..'9'] then
                    textsize := (10*textsize) + ord(ch)-ord('0');
              until ch = ' ';
              i := 1;
              while i <= namesize  do
                 begin
                  read(directory,ch);
                  if fname[i] <> ch
                      then begin
                      i := namesize + 2;
                      index := index + 1;
                      end
                      else i := i + 1;
                  end;


               if i = namesize +1 then found := true
                      else if not eof(directory) then
                        readln(directory);
               end;
             if i = namesize + 1
                   then begin
                   writeln(textsize :1,' lines in file ',fname);
              reset(directory);
```

```
               rewrite(tempd);
               for i := 1 to index do
                       begin

{write the contents of the directory, minus the
file just opened to the scratch file}

                       repeat
                         read(directory,ch);
                         write(tempd,ch);
                       until eoln(directory);
                       readln(directory);
                       writeln(tempd);
                       end;
                    repeat
                      read(directory,ch) until eoln(directory);
                    readln(directory);
                    while not eof(directory) do
                       begin
                       while not eoln(directory) do
                           begin
                           read(directory,ch);
                           write(tempd,ch);
                           end;
                        readln(directory);
                        writeln(tempd);
                        end;
                   k := 1;
                   if textsize > 0 then

{read the first 40 lines of text into the buffer}

                   readbuf(k);
                   end
                   else begin
                   writeln('-creating file ',fname,'-');
                   textsize := 0;
                   rewrite(tempd);
                   reset(directory);
                   while not eof(directory) do
                        begin
                        while not eoln(directory) do
                              begin
                              read(directory,ch);
                              write(tempd,ch);
                              end;
                        readln(directory);
                        writeln(tempd);
                        end;
                   rewrite(afile,fname);
```

```
{update the buffer status record and free list}

                head.ptr := nill;
                head.firstline := 0;
                free := 1;
                for j := 1 to bufsize - 1 do
                   buff[j].nextline := j + 1;
                buff[bufsize].nextline := nill;
                end;
            fileopen := true;
            end
            else begin
                 writeln('-incorrect file name-');
            end;

        end ;       { getfile }
```

```
{
****************************************************************
                              PROCEDURE CLOSE
****************************************************************

PURPOSE:  This procedure closes a previously opened text
          file.  The contents of the scratch file <tempd>
          are read into the SLED directory <directory>
          The text size of the open file and the file
          name are then entered as the last line of the
          directory.

VARIABLES:
          directory : SLED file directory
          tempd :     scratch file
          fname :     currently open file name
          textsize :  number of lines in currently open file
          digit :     character representation of integer textsize

****************************************************************}

     procedure close;

     var ch : char;
         digit, i, j : integer;
         number : array [1..5] of char;

     begin

            for j := 1 to 5 do
                number[j] := ' ';
            writeln('-closing file ',fname,'-');
            j := 5;

{convert integer textsize to ASCII representation}

           if textsize = 0 then number[j] := '0';
           while textsize <> 0 do
                begin
                digit := textsize mod 10;
                digit := digit + ord('0');;
                textsize := textsize div 10;
                ch := chr(digit);
                number[j] := ch;
                j := j -1;
                end;

{insert text size into scratch file}

          for j := 1 to 5 do
```

```
                      if number[i] <> ' ' then
                         write(tempd,number[i]);
                 write(tempd,' ');

{insert file name into scratch file}

                 for i := 1 to namesize do
                    write(tempd,iname[i]);
                 writeln(tempd);

{write scratch file to SLED directory}

                 rewrite(directory);
                 reset(tempd);
                 while not eof(tempd) do
                       begin
                       while not eoln(tempd) do
                            begin
                            read(tempd,ch);
                            write(directory,ch);
                            end;
                        readln(tempd);
                        writeln(directory);
                        end;
      writebuf;

      end ;    { close }
```

```
{
**************************************************************
                                           PROCEDURE OPEN
**************************************************************

PURPOSE:  This procedure opens a user inputed text file.
          If a file is currently open, it is closed and
          the new file then opened.  If the file name is
          not specified, a user prompt is generated.  The
          file is then opened by procedure getfile.

VARIABLES:
          inv :  true if command is invalid

*************************************************************}




     procedure open;

     var  com : char;
          inv : boolean;

     begin

{close a previously opened file}

     if fileopen
          then begin
          close;
          fileopen := false;
          end;
     rewrite(tempd);
     com := ' ';
     inv := false;
     while not eoln(input) and (com <> ct) do
          begin
          read(com);
           if com <> ct
               then begin
               if not eoln(input) then repeat read(com) until
                    eoln(input);
               writeln(errmsg);
               errcnt := errcnt + 1;
               inv := true;
               end;
```

```
         end;
         if not inv

{if no file name, prompt user}

              then begin
              if eoln(input)
                   then begin
                   write('filename?>');
                   readln;
                   getfile;
                   end
                   else getfile;
               end;
      if eoln(input)
            then begin
            readln;
            write('t>');
            end;
     errcnt := 0;

     end ;     { open }
```

```
{
****************************************************************
                                        PROCEDURE QUIT
****************************************************************

PURPOSE:  This procedure closes the currently open file if one
          exists by calling procedure close.

VARIABLES:
          inv : true if command is invalid

****************************************************************}


     procedure quit;

     var  ch : char;
          inv : boolean;

     begin

     ch := ' ';
     inv := false;
     while not eoln(input) and (ch <> ct) do
        begin
        read(ch);
        if ch <> ct
           then begin
           if not eoln(input) then repeat read(ch) until
                  eoln(input);
           write(errmsg);
           errcnt := errcnt + 1;
           inv := true;
           end;
      end;
      if not inv and fileopen then begin
          close;
          fileopen := false;
          errcnt := 0;
          end;
       if eoln(input) then begin
          readln;
          end;

      end ;     { quit }
```

```
{
*****************************************************************
                                        PROCEDURF WRITEMENU
*****************************************************************

        PURPOSE:        This procedure provides the SLFD user with a descrip-
                                tion of the various commands available in SLED.  It can
                                be called by the user typing "M" or is automatically
                                called if two invalid commands in a row are sub-
                                mitted.  The procedure functions by utilizing a
                                file "menu" which contains the SLED command summary.
                                The file is reset, a character is read, then written
                                etc. until eoln is reached.  Then skip down to the
                                next line of "menu" write the line just read and rep-
                                eat the process until eof is reached.

                                        A copy of the command menu is included in the
                                program documentation.

******************************************************************}

procedure writemenu;

begin

        reset(menu);
    while not eof(menu) do begin
             while not eoln(menu) do begin
               read(menu,ch);
               write(ch)
         end;
         readln(menu);
          writeln
       end;
               if not cmderror then begin
         if not eoln(input) then
                repeat read(ch) until (ch = ct) or eoln(input);
             if eoln(input) then begin
                    readln;
                   write('E>');
                    end;
                 end
                 else write('E>');

end ;       { writemenu }
```

```
{
*****************************************************************
                                  PROCEDURE  WRITEVERS
*****************************************************************

        PURPOSE:        This procedure provides the SLED user with a des-
                                cription of the program version.  In SLED it can be
                                called by the user typing a "V".  The procedure
                                functions by utilizing a file "version" which
                                contains the version documentation.  The file is
                                reset, a character is read then written etc. until
                                eoln is reached.  Then skip down to the next line
                                of "version", write the line just read and repeat the
                                process until eof is reached.

                                        A copy of the version format is included in the
                                program documentation.

********************************************************************
}



 procedure writevers;



 begin

         reset(version);
         while not eof(version) do begin
                  while not eoln(version) do begin
                             read(version,ch);
                             write(ch)
                  end;
                  readln(version);
          writeln;
          end;
               if not eoln(input) then repeat
               read(ch) until (ch = ct) or eoln(input);
            if eoln (input) then begin
                  readln;
                  write('!>');
                  end;

 end ;      { writevers }
```

```
{
*****************************************************************
                                              PROCEDURE CHANGETERM
*****************************************************************

        PURPOSE: This procedure is used to change the value of the
                 logical terminator from its optional value of $.
                 This is done by inputing the new value of the log-
                 ical terminator from the console and assigning it to
                 the variable "ct". At the same time, the 50 line
                 buffer is scanned and the logical terminators
                 within each line of the buffer are  scanned
                 and changed to the new value.

VARIABLES :
                 Variables i and j are used as counters in repetitive
                 statements.  Subj is used as a temporary holder of
                 the new logical terminator.


*****************************************************************}



procedure changeterm;

                var  i:integer;
                     j:integer;
                     subj:char;

 begin

        writeln;
        writeln('ENTER THE CHARACTER DESIRED');
        writeln('AS A NEW LOGICAL MESSAGE TERMINATOR');
        writeln('MUST BE A PRINTABLE CHARACTER');
        writeln('CURRENT TERMINATOR IS  ',ct );
        writeln;
                                readln;
                                write('I>');
        read(letter);
        for i := 1 to bufsize do
        begin
        for i := 1 to linesize do
                begin
                subj := buff[i].linestring[j];
                if subj = ct then begin
                                                        buff[i].linestring[j] := letter
                                                end;
                end
```

```
          end;
 ct := letter;
  writeln('LOGICAL TERMINATOR CHANGED TO  ',ct);
readln;
write('E>');

 end; { changeterm }
```

```
{
**********************************************************************
                                             PROCEDURE ERRORUUTINE
**********************************************************************

PURPOSE : This procedure is called whenever there are two consec-
-------     utive input mistakes made by the operator. The command
                  menu will be printed on the CRT screen to inform the
                  user of the proper procedures for data display and
                  input. The errorcount is reset to zero whenever this
                  procedure is called ( errcnt is a global variable ).
                  when a proper command is entered by the operator, the
                  errorcount is also set to zero. This parameter can be changed
                  to display the command menu less frequently by a small change
                  in the main program.

********************************************************************
}


 procedure errroutine;

 begin
      writemenu;
      errcnt := 0;
 end; { errroutine }
```

```
{*******************************************************************
                              MAIN PROGRAM
********************************************************************

  PURPOSE:  The main program is utilized to select the appropriate
                      riate text editing procedure utilizing the inputs
                      from the console.  It also performs the initial
                      editing of these inputs.  The program will read the
                      first 1 or 2 letters input to the console (depending
                      on command input) and determine which procedure must
                      be entered to execute the desired command.  This is
                      accomplished with a sequential scan of a series of
                      if statements in which the first letter and if
                      necessary the second letter of the command input to
                      the console is checked against the authorized initial
                      letter or letters of SLED commands.  If the
                      initial editing is successfully completed in one of
                      the if statements, the procedure associated with
                      that if statement is entered.  The remainder of the
                      command is edited in the called procedure.  If the
                      sequential scan gets through the last if
                      statement and no match is found with a valid command,
                      the following is done:



                        1. Error message printed

                          2. Increment error counter

                          3. Check value of the error counter;
                             if large enough enter procedure to write command
                                menu

                          4. Check if more commands follow; if not, read a line

                          5. Return to statement 10 in the main program
                                and read the next command.

                      In each if statement associated with commands which
                      would have follow-on parameters after the initial
                      letter(s), the status of the error count is checked to
                      determine the success of editing in the procedure
                      and if necessary the error message or menu is printed.
                      After every return from a procedure, the program goes to
                      statement 10 and reads  the next command except
                      for the quit command which causes the program
                      to terminate.
VARIABLES : The following global variables are used in the main
                       program :
                               errcnt : count of user generated input errors
                               cmderror : boolean used to check for proper input
                                                 commands
                               fileopen : boolean check to ensure that a file is
                                                 opened before allowing the proceaures
                                                 to operate
                               latter : character input of the user


****************************************************************)


 begin

     ct := '$';
     fileopen := false;
        writeln;
        writeln;
        writeln('SLED');
        writeln;
        writeln('HELP INFO TYPE "M" FOR COMMAND MENU');
        writeln('OR "V" FOR SYSTEM VERSION');
        writeln;
        errcnt := 0;
   cmderror := true;
        write('E>');

10:  read(letter);


               if (letter = 'L')   then begin if fileopen then displayline
                              else begin writeln('-no file open-');
                                         readln;
                                         write('F>');
                                           end;
                                                                     if errcnt > 1 then errroutine;
                                                                     goto 10
                                                   end;
               if  (letter = 'S')   then begin if fileopen then screenline
                               else begin writeln('-no file open-');
                               readln;
                                write('F>');
                                end;
                                                                 if errcnt > 1 then errroutine;
                                                            goto 10
                                                       end;
               if letter = 'Z' then begin cmderror := false;
```

```
              writemenu;
                  cmderror := true;
                                                    errcnt := 0;
                                                    goto 10
                                          end;
if letter = 'V' then begin writevers;
                                                    errcnt := 0;
                                                    goto 10
                                          end;
if letter = 'O' then begin open;
                                                     errcnt := 0;
                                                    goto 10
                                          end;
if letter = 'Q' then begin quit;  goto 100;
                end;
if letter = 'A' then
        begin
                read(letter);

                if (letter = 'L')   then begin if fileopen then appendcom
                          else begin
                            writeln('-no file open-');
                             readln;
                             write('E>');
                              end;
                                                  if errcnt > 1 then errroutine;
                                                                        goto 10
                                                           end
                   else begin
                   errcnt := errcnt + 1;
                  writeln(errmsg);
                  if errcnt > 1 then errroutine
                      else write('E>');
                        readln;

                                                     goto 10
                                                            end
        end;
if letter = 'D' then
        begin
                read(letter);
                if (letter= 'S')   then begin if fileopen then dispcom
                       else begin
                       writeln('-no file open-');
                       readln;
                       write('E>');
                       end;
                                                                  if errcnt > 1 then errrout
                                                                          goto 10
                                                            end
                  else begin
                  errcnt := errcnt + 1;
                 writeln(errmsg);
                   if errcnt > 1 then errroutine
                     else write('E>');
                         readln;
                                                                         goto 10
                                                                 end
        end;
if letter = 'C' then
        begin
                read(letter);
                if (letter = 'T')   then begin changeterm;
                                                                    errcnt := 0;
                                                                    goto 10
                                                           end
                   else begin
                    errcnt := errcnt + 1;
                    writeln(errmsg);
                   if errcnt > 1 then errroutine
                     else write('E>');
                          readln;
                                                                    goto 10
                                                           end
        end;
if letter = 'R' then
        begin
                read(letter);
                if (letter = 'L')   then begin if fileopen then replcom
                   else begin
                   writeln('-no file open-');
                   readln;
                   write('F>');
                   end;
                                                         if errcnt > 1 then errroutine;
                                                                    goto 10
                                                                 end;
                if (letter = 'S')   then begin if fileopen then strepcom
                  else begin
                  writeln('-no file open');
                  readln;
                   write('E>');
                   end;
                                                         if errcnt > 1 then errroutine;
                                                                         goto 10
                                                           end
                 else begin
                  errcnt := errcnt + 1;
                      writeln(errmsg);
                   if errcnt > 1 then errroutine
```

```
                                                else write('E>');
                                                        readln;
                                                                                                       goto 10
                                                                                                 end
                                       end;
        errcnt := errcnt + 1;
         writeln(errmsg);
                   if errcnt > 1 then errroutine
                       else write('F>');
                             if letter <> ' ' then readln;
           goto 10;

100:  writeln(' ');

end.     { main program }
```

```
(************************ END SLED ********************************)
```

NON STANDARD PASCAL USES IN THIS VERSION OF SLED
Obtained from the UC Berkeley PASCAL traslator (pi -s option)

s 240 - Two argument forms of reset and rewrite are non-standard

s 297 - Two argument forms of reset and rewrite are non-standard

s 1920 - Two argument froms of reset and rewrite are non-standard

s 172 - Two argument forms of reset and rewrite are non-standard

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Office of Research Administration Code 012A Naval Postgraduate School Monterey, California 93940	1
4.	Chairman, Code 52Bz Computer Science Department Naval Postgraduate School Monterey, California 93940	30
5.	Lyle A. Cox, Jr., Code 52Cl Computer Science Department Naval Postgraduate School Monterey, California 93940	10
6.	I. Larry Avrunin David Taylor Naval Ship Research and Development Center (Code 18) Carderock Laboratory Bethesda, MD 20084	1
7.	R. P. Crabb, Code 9134 Naval Ocean Systems Center San Diego, CA 92152	3
8.	G. H. Gleissner David Taylor Naval Ship Research and Development Center (Code 18) Carderock Laboratory Bethesda, MD 20084	1
9.	Kathryn Heninger, Code 7503 Naval Research Lab Washington D. C. 20375	3
10.	Ronald P. Kasik, Code 4451 Naval Underwater Systems Center Newport, RI 02840	3

11. Commander, Code 503 3
Naval Sir Development Center
Warminster, Pennsylvania 18974

12. Mark Underwood, Code P204 3
NPRDC
San Diego, California 92152

13. Michael Wallace, Code 1828 3
DTNSRDC
Bethesda, MD 20084

14. Walter P. Warner, Code K70 3
NSWC
Dahlgren, Virginia 22448

15. John Zenor, Code 31302 3
Naval Weapons Center
China Lake, California 96555

ATE

ME